AnonProt: Optimizing Block Period and Commit Latency in Chain-Based Rotating Leader BFT

Anonymous Author(s)

***Abstract*—Existing chain-based BFT SMR protocols for the partially synchronous network model with a constant minimum block commit latency incur a minimum block period of 2δ (where δ is the message transmission latency) between the proposals of distinct consecutive honest leaders. While a protocol with a minimum block period of δ exists in the synchronous model, its minimum commit latency is linear in the size of the system.**

To close this gap, we present the first chain-based BFT SMR protocols with a minimum delay of δ between the proposals of distinct consecutive honest leaders and a minimum block commit latency of 3δ. We present three different protocols for the partially synchronous network model under different notions of optimistic responsiveness, two of which implement pipelining and one of which does not. All of our protocols achieve reorg-resilience and have minimal view-length; properties that many existing chain-based BFT SMR protocols lack. We evaluated our protocols in networks of up to 200 nodes wherein they achieved average increases of 51%–53% in block throughput and decreases of 43%–54% in block commit latency compared to the state-of-the-art chained-based protocol, Jolteon, in their happy paths. Our non-pipelined protocol also performed particularly well under failures, delivering 74% higher throughput and 80% lower latency than Jolteon under Jolteon's best-case leader schedule in a 50 node network with 16 failures, and 412% higher throughput and 99.3% lower latency under its worst-case schedule.

I. Introduction

Blockchain networks have become increasingly popular as mechanisms for facilitating decentralised, immutable and verifiable computation and storage. These networks leverage Byzantine fault-tolerant (BFT) consensus protocols to ensure that their participants (called *nodes*) execute the same sequence of operations (called *transactions*), despite some of them exhibiting arbitrary failures. Many blockchain networks also prioritize *fairness*; i.e., they strive to ensure that i) client transactions are processed promptly, without granting any client an unfair advantage over the others, and ii) nodes have an equal opportunity to be rewarded for the work that they do in the system. Public blockchain networks in particular also tend to be large, supporting hundreds (e.g. [24]) or thousands (e.g. [7]) of nodes in the pursuit of decentralization, and aim to cater to many concurrent clients. Accordingly, the consensus protocols driving these networks need to be efficient, maximising transaction throughput and minimising end-to-end commit latency (i.e., the time between a client submitting a transaction and it being executed by the blockchain).

To these ends, prior works [29], [33], [18], [4], [12], [19], [25] have leveraged two key strategies: i) *block chaining*, and; ii) frequent *leader rotation*. In the block chaining (or *chained*) paradigm, transactions are grouped into *blocks* that explicitly reference one or more existing blocks (called the *parents* of the block), typically by including their hashes. This enables an optimization called *pipelining*, wherein the *vote* acknowledgement messages sent by the nodes in the course of agreeing upon one block can be counted towards the finalization of its parents, reducing the communication and computational complexity of the protocol by a constant factor. Our work focuses on the *chain-based* subcategory of chained protocols, wherein each block has exactly one parent, as opposed to DAG-based protocols in which a block may have many parents. In rotating-leader chain-based protocols, the leader responsible for proposing these blocks is changed at regular intervals, even when functioning correctly. This helps to fairly distribute the proposal workload and any related rewards. Additionally, the more frequently leaders are rotated the less amount of time a Byzantine (faulty) leader has to manipulate the ordering of pending transactions, improving censorship resistance. Accordingly, rotating-leader protocols often rotate the leader after every block proposal, an approach called *leader-speaks-once* (LSO). This paper seeks to optimize chain-based BFT consensus performance in a modified version of the LSO setting, which we name *leader-certifies-one* (LCO). Whereas an LSO protocol allows a leader to propose only a single block, an LCO protocol allows it to propose multiple but ensures that it produces no more than one certified block during its tenure. Even as the previously cited works need not be implemented as LSO, our protocols need not be implemented as LCO, however, it is in this setting that they have the greatest advantage. Going forward, we refer to chain-based BFT consensus protocols that implement leader rotation as *CRL protocols*, and to their specific LSO or LCO variants as *CLSO protocols* and *CLCO protocols*, respectively.

Our work targets the *partially synchronous* network model [17] wherein there exists a time called the *Global Stabilization Time* (GST) after which message delivery takes at most Δ time. We use δ to denote the actual delivery time, which naturally satisfies $\delta \leq \Delta$ after GST. Many recent CRL protocols for this setting have focused on reducing communication complexity. Some have achieved linear communication complexity in their *steady state* phases [21], [18] (i.e. when the protocol makes progress under a fixed leader), while others obtain this result in their *view-change* phases [33], [25] (i.e. when the protocol elects a new leader) as well. However, these protocols sacrifice efficiency in several important metrics in their pursuit of linearity, including i) *minimum commit latency* (i.e., the minimum delay between a block being proposed and it being committed by all honest—i.e., non-faulty—nodes), ii) *minimum view change block period* (i.e., the minimum delay

TABLE I: **Theoretical comparison of chained rotating leader BFT SMR protocols, after GST**

		Model	Minimum Commit Latency	Minimum View Change Block Period	Reorg Resilience	View Length	Pipelined	Communication Complexity		Optimistic Responsiveness	
								steady-state	view-change	standard	consecutive honest
HotStuff	[33]	psync.	$7\delta^*$	2δ	✗	4Δ	✓	$O(n)$	$O(n)$	✓	✓
Fast HotStuff	[21]	psync.	5δ	2δ	✗	4Δ	✓	$O(n)$	$O(n^2)$	✓	✓
Jolteon	[18]	psync.	5δ	2δ	✗	4Δ	✓	$O(n)$	$O(n^2)$	✓	✓
HotStuff-2	[25]	psync.	5δ	2δ	✓	7Δ	✓	$O(n)$	$O(n)$	✗	✓
PaLa	[15]	psync.	4δ	2δ	✓	5Δ	✓	$O(n^2)$	$O(n^2)$	✗	✓
ICC	[12]	psync.	3δ	2δ	✗	4Δ	✗	$O(n^2)$	$O(n^2)$	✗	✓
Simplex	[14]	psync.	3δ	2δ	✓	3Δ	✗	Unbounded$^+$	$O(n^2)$	✗**	✓
Apollo	[6]	sync.	$(f+1)\delta$	δ	✓	4Δ	✗	$O(n)$	$O(n^2)$	✗	✓
This work (§III)		psync.	3δ	δ	✓	5Δ	✓	$O(n^2)$	$O(n^2)$	✗	✓
This work (§IV)		psync.	3δ	δ	✓	3Δ	✓	$O(n^2)$	$O(n^2)$	✓	✓
This work (§V)		psync.	3δ	δ	✓	3Δ	✗	$O(n^2)$	$O(n^2)$	✓	✓

*The 7δ good-case latency is assuming the next leader aggregates the vote messages; in the original HotStuff specification, the current leader aggregates the vote messages and forwards it to the next leader, incurring an additional 3δ. $^+$Simplex [14] requires the entire blockchain be included while sending a proposal which eventually requires unbounded communication. **Additionally, Simplex [14] claims responsiveness only when all nodes are honest.

between the proposals of different honest leaders), and iii) *view length* (i.e., the duration a node waits in a view before it considers the current leader to have failed). In particular, these works require at least 5δ to commit a new block, at least 2δ between honest proposals in the LSO setting, and view lengths of at least 4Δ. Moreover, since these protocols all rely on a designated node to aggregate vote messages and forward the resulting certificates, they grant the adversary the power to censor certificates for honest proposals when this aggregator is Byzantine—even after GST. Accordingly, any implementation of these protocols that uses any node other than the original proposer as the vote aggregator is not *reorg resilient*; i.e., it cannot guarantee that an honest leader that proposes after GST will produce a block that becomes a part of the committed blockchain.

A recent line of works [12], [14] designed CRL protocols with minimum commit latencies of 3δ. However, these protocols are in the non-pipelined setting, have minimum view change block periods of 2δ and either have long view lengths [12] or are less practical in nature [14]. To the best of our knowledge, Apollo [6] is the only existing CRL protocol with a minimum view change block period of δ. However, it incurs a minimum commit latency of $(f+1)\delta$ even during failure-free executions and assumes a synchronous network. As far as we know, no chain-based consensus protocol has simultaneously achieved a minimum view change block period of δ and a constant commit latency. To close this gap, our paper explores the design of such protocols.

We first present two consensus protocols in the form of state machine replication (SMR) protocols for the pipelined setting, each of which satisfies a different notion of *responsiveness*: i) *optimistic responsiveness* [33] (Definition 6) and ii) *optimistic responsiveness under consecutive honest leaders* [19] (Definition 7). Informally, the former requires an honest leader to make progress in $O(\delta)$ time after GST (i.e., without waiting for $\Omega(\Delta)$ time) while the latter requires an honest leader to make progress in $O(\delta)$ time only when the previous leader is also honest. Our first protocol satisfies the former definition and is simpler to reason about, but has a longer view length. The second satisfies the latter definition and has a shorter view length, but is more complex.

Both of our protocols require only two consecutive honest leaders after GST to commit a new block, and achieve reorg resilience through vote-multicasting. This strategy, together with an optimization that we call *optimistic proposal*, also enables them to achieve both a minimum view change block period of δ and a minimum commit latency of 3δ. We say that a protocol implements optimistic proposal if a leader is allowed to "optimistically" extend a block proposed by its predecessor without waiting to observe its certification. We implement this in our protocols by allowing the leader of the next view to propose a new block when it votes for a block made by the leader of the current view. Accordingly, we obtain the following result:

Theorem 1. *There exists a pipelined CRL protocol tolerating $f < n/3$ Byzantine faults with a minimum view change block period of δ and a minimum commit latency of 3δ.*

As we previously mentioned, pipelining reduces communication and computational overhead of the protocol by a constant factor. However, although this gives pipelined protocols good latency when all messages require a similar amount of time to propagate and process, pipelining actually increases commit latency when blocks take sufficiently longer to propagate or process than votes. Accordingly, we also present a non-pipelined variant of our second protocol in Section V, giving us the following result:

Theorem 2. *There exists a non-pipelined CRL protocol tolerating $f < n/3$ Byzantine faults with a minimum view change block period of δ and a minimum commit latency of 3δ.*

This final protocol retains standard optimistic responsiveness and requires only a single honest leader to commit a new block after GST.

Evaluation and Related Work. We evaluated LCO implementations of all three of our protocols against an LSO implementation of Jolteon, all three of which delivered superior performance in both the happy path and in the presence of

failures. We present the corresponding results in Section VI, and follow this with a more detailed discussion of related work in Section VII.

II. Preliminaries

We consider a system comprised of a set $\mathcal{V} = (P_1, \ldots, P_n)$ of n nodes running a protocol $\mathcal{P}$ in a reliable, authenticated all-to-all network. We assume the existence of a static adversary that may corrupt up to $f < n/3$ of the nodes when $\mathcal{P}$ begins, which it may then cause to behave arbitrarily. We refer to all nodes under the control of the adversary as being *Byzantine*, while we refer to those that adhere to $\mathcal{P}$ as being *honest*. We define a *quorum* as a set of $\lfloor \frac{n}{2} \rfloor + f + 1$ nodes. Henceforth, for the sake of simplicity, we assume that $n = 3f + 1$ and that a quorum therefore contains $2f + 1$ nodes.

We consider the partial synchrony model of Dwork et al. [17]. Under this model, the network starts in an initial state of asynchrony during which the adversary may arbitrarily delay messages sent by honest nodes. However, after an unknown time called the *Global Stabilization Time* (GST), the adversary must ensure that all messages sent by honest nodes are delivered to their intended recipients within Δ time of being sent. In our initial analyses, we use δ to characterize the actual (variable) transmission latencies of messages of all types after GST and observe that $\delta \leq \Delta$. However, in our later analyses we update our model to the *modified partially synchronous model* of Blum et al. [8], wherein small messages (such as votes) are delivered within ρ time after GST while large messages (such as block proposals) are delivered within Λ time such that $\rho < \Lambda$.

We make use of digital signatures and a public-key infrastructure (PKI) to prevent spoofing and replay attacks and to validate messages. We use $\langle x \rangle_i$ to denote a message x digitally signed by node P_i using its private key. In addition, we use $\langle x \rangle$ to denote an unsigned message x sent via an authenticated channel. We use $H(x)$ to denote the invocation of the hash function H with input x.

A. Property Definitions

State Machine Replication. A state machine replication (SMR) protocol run by a network $\mathcal{V}$ of n nodes receives requests (transactions) from external parties, called *clients*, as input, and outputs a totally ordered log of these requests. We recall the definition of SMR given in [2], below.

Definition 1 (Byzantine Fault-Tolerant State Machine Replication [2])**.** *A Byzantine fault-tolerant state machine replication protocol commits client requests as a linearizable log to provide a consistent view of the log akin to a single non-faulty node, providing the following two guarantees.*

- ***Safety****. Honest nodes do not commit different values at the same log position.*
- ***Liveness****. Each client request is eventually committed by all honest nodes.*

Definition 2 (Minimum View Change Block Period (ω))**.** *The minimum view change block period ω of a chained consensus protocol $\mathcal{P}$ is the minimum latency between the proposal of a block B by an honest node P_i and its extension (directly or indirectly) by any honest node P_j such that $P_j \neq P_i$.*

Definition 3 (Minimum Commit Latency (λ))**.** *A consensus protocol has a minimum commit latency of λ if all honest nodes that commit a block proposed at time t, do so no earlier than $t + \lambda$.*

In this paper, we measure the above two metrics in relation to message transmission latency and assume that message processing time is relatively negligible.

Definition 4 (View Length (τ))**.** *A consensus protocol has a view length of τ if an honest node that enters view v at time t considers the view to have failed if it remains in v until $t + \tau$.*

Definition 5 (Reorg Resilience)**.** *We say that a consensus protocol is reorg resilient if it ensures that when an honest leader proposes after GST, one of its proposals becomes certified and this proposal is extended by every subsequently certified proposal.*

Optimistic Responsiveness. Responsiveness requires a consensus protocol to make progress in time proportional to the actual network delay (δ) and independent of any known upper bound delay (Δ) when a leader is honest [28]. Optimistic responsiveness requires this same guarantee, but only when certain optimistic conditions hold. Several variations [33], [4], [19], [14] have been formulated in the literature, two of which we make use of in this paper and recall below.

Definition 6 (Optimistic Responsiveness [33])**.** *After GST, any correct leader, once designated, needs to wait just for the first $n - f$ responses to guarantee that it can create a proposal that will make progress. This includes the case where a leader is replaced.*

We note that in [33], the term "make progress" means that all honest nodes will vote for the correct (honest) leader's proposal, not that all honest nodes observe a certificate for the included block; i.e. optimistic responsiveness does not imply reorg resilience. We also clarify that for LSO protocols, these (at most) $n - f$ responses should be messages from the previous view.

Definition 7 (Optimistic Responsiveness (Consecutive Honest) [19], [4])**.** *We say a protocol is optimistically responsive (consecutive honest) if after GST, for any two honest leaders, L_i, L_j for views $i < j$ respectively, L_j sends its proposal within $O(\delta)$ time after view i finishes.*

Importantly, this variant of optimistic responsiveness allows the protocol to wait for $\Omega(\Delta)$ time before proposing in the new view when the leader of the previous view is Byzantine.

B. Protocol Definitions

View-based execution. Our protocols progress through a sequence of numbered *views*, with all nodes starting in view 1 and progressing to higher views as the protocol continues.

Each view v is coordinated by a designated leader node L_v that is responsible for proposing a new block for addition to the blockchain. For the sake of liveness, we require that the leader election function L continually elects sequences of leaders that contain at least two consecutive (not necessarily distinct) honest leaders after GST for our pipelined protocols, and only one such leader for our non-pipelined protocol. We note that L must additionally change the leader every view for LCO implementations, and must elect each node with equal probability in fair implementations.

Blocks. The blockchains of each of our protocols are initialized with a *genesis block* B_0 that is known to all nodes at the beginning of the protocol. Each block references its immediate predecessor in the chain, which we refer to as its *parent*, with the parent of the genesis block being $\perp$. We say that a block *directly extends* its parent and *indirectly extends* its other predecessors in the chain. For simplicity when reasoning, we also say that a block extends itself. We refer to the predecessors of a given block as its *ancestors* and measure its *height* by counting its ancestors. A block B_k with height k has the format, $B_k := (b_v, H(B_{k-1}))$ where b_v is a fixed payload for the view v for which B_k is proposed, B_{k-1} is the parent of B_k, and $H(B_{k-1})$ is the hash digest of B_{k-1}. We allow the implementation to dictate the contents of b_v (e.g. transactions or hashes of batches of transactions). Accordingly, B_k is *valid* if i) its parent is valid, or if $k = 0$ and its parent is $\perp$, and ii) b_v satisfies the implementation-specific validity conditions. Finally, we say that two blocks B_k and $B'_{k'}$ proposed for the same view *equivocate* one another if they do not both have the same parent and payload.

Block certificates. In our protocols, a node sends a signed vote message to indicate its acceptance of a block. A block certificate $\mathcal{C}_v(B_k)$ for view v consists of a quorum of distinct signed vote messages for B_k for v. We use $\mathcal{C}_v$ to denote a block certificate for view v when knowledge of the related block is irrelevant to the context. We rank block certificates by their view such that $\mathcal{C}_v \leq \mathcal{C}_{v'}$ if $v \leq v'$. We provide more detailed definitions in the following sections where necessary.

Timeout messages and timeout certificates. Our protocols maintain the liveness SMR property by requiring nodes to request a new leader when they fail to observe progress in their current views after a certain amount of time. They do so by sending signed timeout messages for the view, the contents of which are protocol-specific. A view v timeout certificate, denoted $\mathcal{TC}_v$, consists of a quorum of distinct signed timeout messages for v, denoted $\mathcal{T}_v$.

III. Simple AnonProt

We now present Simple AnonProt (Figure 1), the first of our CRL protocols for the pipelined setting. Simple AnonProt achieves $\omega = \delta$, $\lambda = 3\delta$, reorg-resilience and responsiveness under consecutive honest leaders. We first discuss how our protocols obtain the former properties before elaborating on Simple AnonProt itself.

Towards achieving $\omega = \delta$ and $\lambda = 3\delta$. Prior CRL protocols require L_v to observe C_{v-1} before proposing during their happy paths (i.e. when views progress without any honest node sending a timeout message—as opposed to the *fallback path*). This is intended to help honest leaders create blocks that will become committed, but is unnecessarily strict for this purpose and naturally affects $\omega \geq 2\delta$ and $\lambda \geq 4\delta$ in the pipelined setting. Our protocols improve upon these results by requiring i) the leader of view v to propose a block for v, say B_k, upon voting for a block in $v-1$, say B_{k-1}, and; ii) nodes to multicast their votes. Allowing leaders to propose optimistically in this way enables voting for B_{k-1} to proceed in parallel with the proposal of B_k. Moreover, when the dissemination times of vote and proposal messages are equal, having nodes multicast their votes ensures that if all honest nodes vote for B_{k-1} then they will all receive B_k at the time that they construct $\mathcal{C}_{v-1}(B_{k-1})$, allowing them to vote for B_k and L_{v+1} to propose immediately upon entering v. Hence, in the happy path, L_{v+1} proposes as soon as it receives L_v's proposal, giving our protocols an ω of δ. Furthermore, since our pipelined protocols require two consecutive views to produce certified blocks before a new block can be committed, requirements (i) and (ii) also give our protocols a λ of 3δ.

A. Protocol Details

We define Simple AnonProt in Figure 1 as a series of event handlers to be run by each node $P_i \in \mathcal{V}$. We elaborate on these below.

Advance View and Timeout. P_i enters view v from some view $v' < v$ upon receiving a view $v-1$ block certificate or a view $v-1$ timeout certificate (i.e. P_i never decreases its local view). Before doing so, it first multicasts this certificate. This ensures that if the first honest node enters v after GST then all honest nodes will enter v or higher within Δ thereafter, helping our protocol to obtain liveness and reorg resilience. Subsequently, P_i updates lock_i to the highest ranked block certificate that it has received so far and if lock_i is not $\mathcal{C}_{v-1}$ then P_i unicasts a status message containing lock_i to L_v. We note that P_i only updates lock_i during the view transition process and does not do so after entering the new view, even if it receives a higher ranked block certificate. This ensures that the block certificate reported in a status message corresponds to its honest sender's lock_i for the duration of v, meaning that if L_v waits to receive status messages from all honest nodes before proposing, then it is guaranteed to extend the block certified by the highest ranked block certificate locked by any honest node. Finally, P_i enters v, resets $\mathsf{view\text{-}timer}_i$ to 5Δ and starts counting down. If L_v is honest and the network is synchronous then P_i should enter $v+1$ within 5Δ of entering v. If it does not, then it considers the current leader to have failed and so multicasts $\langle \mathsf{timeout}, v \rangle_i$ to request a view change and prevent the protocol from halting. P_i also does this whenever it observes that at least one other honest node has requested a view change for v.

Propose. Simple AnonProt allows two proposals to be created during view v: i) an optimistic proposal for view $v+1$, and; ii) a normal proposal for v. In the former case, L_{v+1} multicasts $\langle \mathsf{opt\text{-}propose}, B_{k+1}, v+1 \rangle$, where B_{k+1} extends

A Simple AnonProt node P_i runs the following protocol whilst in view v:

1) **Propose.** If P_i is L_v and enters view v at time t, propose: (i) upon receiving $\mathcal{C}_{v-1}(B_{k-1})$ before $t + 2\Delta$, or; (ii) at $t + 2\Delta$. Do so by multicasting $\langle \mathsf{propose}, B_k, \mathcal{C}_{v'}(B_{k-1}), v\rangle$, where $\mathcal{C}_{v'}(B_{k-1})$ is the highest ranked block certificate known to L_v and B_k extends B_{k-1}.
2) **Vote.** P_i votes once using one of the following rules:
   a) Upon receiving the first optimistic proposal $\langle \mathsf{opt\text{-}propose}, B_k, v\rangle$, where B_k extends B_{k-1}, if $\mathsf{lock}_i = \mathcal{C}_{v-1}(B_{k-1})$ then multicast $\langle \mathsf{vote}, H(B_k), v\rangle_i$.
   b) Upon receiving the first normal proposal $\langle \mathsf{propose}, B_k, \mathcal{C}_{v'}(B_h), v\rangle$, if $\mathcal{C}_{v'}(B_h) \geq \mathsf{lock}_i$ and B_k extends B_h then multicast $\langle \mathsf{vote}, H(B_k), v\rangle_i$.
3) **Optimistic Propose.** If P_i is L_{v+1} and votes for B_k in v, multicast $\langle \mathsf{opt\text{-}propose}, B_{k+1}, v + 1\rangle$ where B_{k+1} extends B_k.
4) **Timeout.** Upon receiving $f + 1$ distinct $\langle \mathsf{timeout}, v\rangle_*$ or when $\mathsf{view\text{-}timer}_i$ expires, stop voting in v and multicast $\langle \mathsf{timeout}, v\rangle_i$.
5) **Advance View.** Upon receiving $\mathcal{C}_{v'-1}(B_h)$ or $\mathcal{TC}_{v'-1}$, where $v' > v$, and before executing any other rule, do the following: i) multicast the certificate; ii) update lock_i to the highest ranked block certificate received so far; iii) unicast a status message $\langle \mathsf{status}, v', \mathsf{lock}_i\rangle$ to $L_{v'}$ if lock_i has a view less than $v' - 1$, iv) enter v', and; v) reset $\mathsf{view\text{-}timer}_i$ to 5Δ and start counting down.

P_i additionally performs the following action in any view:

1) **Direct Commit.** Upon receiving $\mathcal{C}_{v-1}(B_{k-1})$ and $\mathcal{C}_v(B_k)$ such that B_k extends B_{k-1}, commit B_{k-1}.
2) **Indirect Commit.** Upon directly committing B_{k-1}, commit all of its uncommitted ancestors.

Fig. 1: The Simple AnonProt Protocol

B_k, upon voting for B_k in v, hoping that B_k will become certified. When the protocol is operating in its happy path after GST, B_k will indeed become certified, enabling voting for consecutive honest proposals to proceed without delay. In the latter case, L_v multicasts $\langle \mathsf{propose}, B_h, \mathcal{C}_{v'}(B_{h-1}), v\rangle$, where B_h extends B_{h-1}, either upon receiving $\mathcal{C}_{v-1}(B_{h-1})$ within 2Δ time of entering v, or after having $\mathcal{C}_{v'}(B_{h-1})$ as its highest block certificate after waiting for 2Δ after entering v. Since messages are delivered within Δ time after GST, this 2Δ wait ensures that L_v will extend the highest certified block locked by any honest node when it proposes after GST, assisting with liveness and reorg resilience. We require L_v to multicast a normal proposal even when it has already multicasted an optimistic proposal to ensure that it always produces a certified block when it proposes after GST. We note that this requirement can be removed from each of our protocols to obtain the corresponding CLSO variant, but doing so naturally sacrifices reorg resilience because the adversary can cause optimistic proposals to fail, even after GST. We discuss how after introducing the remaining protocol rules.

Vote. Simple AnonProt has two rules for voting, at most one of which each node may invoke at most once per view. Firstly, P_i may vote for an optimistic proposal containing B_k proposed for view v and extending B_{k-1}, when locked on $\mathcal{C}_{v-1}(B_{k-1})$. In the best case, P_i receives the optimistic proposal containing B_k and $\mathcal{C}_{v-1}(B_{k-1})$ simultaneously and so votes for B_k immediately upon entering view v. Alternatively, if P_i receives $\langle \mathsf{propose}, B_h, \mathcal{C}_{v'}(B_{h-1}), v\rangle$ and $\mathcal{C}_{v'}(B_{h-1})$ ranks higher than or equal to lock_i and B_h extends B_{h-1}, then it votes for B_h. Importantly, if L_v creates both an optimistic proposal and a normal proposal with the same parent, then since payloads are fixed for a given view, both proposals will contain the same block. This ensures that all honest nodes will vote for the same block, even if they end up using different vote rules.

Commit. Finally, at any time during protocol execution, when an honest node P_i receives $\mathcal{C}_{v-1}(B_{k-1})$ and $\mathcal{C}_v(B_k)$, it commits B_{k-1} and all of its uncommitted ancestors. We say that a node *directly commits* B_k and *indirectly commits* any ancestors that it commits as a result of committing B_k.

B. Analysis

We now provide some discussion on the properties of Simple AnonProt, including brief intuitions for its safety, liveness and reorg resilience. We are unable to provide full proofs for our protocols in this paper due to space limitations, but the interested reader can find them in [5] (currently anonymized).

How does our protocol achieve safety? The vote and commit rules together ensure that Simple AnonProt satisfies the safety property of SMR. Specifically, if an honest node commits B_k for view v after receiving $\mathcal{C}_v(B_k)$ and $\mathcal{C}_{v+1}(B_{k+1})$, then a majority of the honest nodes must have voted for B_{k+1} in view $v + 1$. Therefore, since honest nodes vote at most once per view, an equivocating $\mathcal{C}_v$ cannot exist so no honest node will be able to commit any block other than B_k for view v. Moreover, since the set of honest nodes that voted for B_{k+1}, say H, must have had $\mathcal{C}_v(B_k)$ when they voted for B_{k+1}, they will either lock this certificate or one of a higher rank upon transitioning from $v + 1$ to a higher view. Once again then, since block certificates must contain votes from a majority of the honest nodes, every block certificate for every view greater than v must contain a vote from at least one member of H. Suppose that v' is the first view greater than $v + 1$ to produce a block certificate and let P_i be a member of H that votes towards $\mathcal{C}_{v'}(B_l)$. Importantly, since P_i must lock $\mathcal{C}_v(B_k)$ before voting for B_l and since no higher ranked block certificate than $\mathcal{C}_{v+1}(B_{k+1})$ can exist before it does so, by the vote rules, B_l must directly extend either B_k or B_{k+1}. By extension then, every block certified for a higher view than v must extend B_k. This is sufficient to ensure safety.

Why propose twice? As previously mentioned, we require our leaders to make normal proposals even if they have already made an optimistic proposal because the adversary can cause optimistic proposals to fail even after GST. Suppose that an honest leader L_v proposes B_k extending B_{k-1} in an optimistic proposal. Per the optimistic vote rule, the adversary can cause

B_k to fail by preventing some honest node from locking $\mathcal{C}_{v-1}(B_{k-1})$. This could happen either due to some other block, say B_l, becoming certified for view $v-1$, or due to the node entering v via $\mathcal{TC}_{v-1}$. In either case, since L_v is guaranteed to observe the highest ranked block certificate locked by any honest node upon entering view v, say $\mathcal{C}_{v'}(B_h)$, before it multicasts its normal proposal, it will be able to multicast a new block, say B_{h+1}, that extends B_h. Therefore, since honest nodes only update their locks when entering a new view, those that receive B_{h+1} whilst in view v will all have $\mathsf{lock}_i \leq \mathcal{C}_{v'}(B_h)$ and hence will vote for it. Thus, this requirement yields two important properties: i) that honest leaders are able to correct themselves when they initially extend a block that fails to become certified, and; ii) that if this block does become certified and its certificate is locked by any honest node, then the block included in the optimistic proposal will become certified even if some honest nodes initially fail to lock this certificate. This ensures that every honest leader that proposes after GST produces exactly one certified block.

How does our protocol achieve reorg resilience and liveness? As we have just explained, Simple AnonProt guarantees that every honest leader that proposes after GST produces exactly one certified block. Suppose that L_v is such an honest leader and produces $\mathcal{C}_v(B_k)$, and let t denote the time that the first honest node enters v. Since the multicasting of block certificates and timeout certificates ensures that all honest nodes will enter view v or higher within $t+\Delta$, if L_v is honest then it will send its last proposal by $t+3\Delta$, so all honest nodes will finish voting before $t+4\Delta$ and thus before any honest node can have sent $\mathcal{T}_v$ or higher. Therefore, either all honest nodes vote for B_k or some honest node must have entered $v+1$ via $\mathcal{C}_v(B_k)$ first. In either case, all honest nodes will receive $\mathcal{C}_v(B_k)$ within 5Δ of the first honest node entering v, so at least $f+1$ honest nodes will lock this certificate. Therefore, since these $f+1$ nodes will not vote for any optimistic proposal that does not directly extend their lock, and since no higher ranked block certificate can exist before they lock $\mathcal{C}_v(B_k)$, every certified block for every view greater than v must extend B_k satisfying Definition 5. Moreover, when L_{v+1} is also honest, it will necessarily be among the $f+1$ honest nodes that lock $\mathcal{C}_v(B_k)$ and will therefore multicast a proposal that extends B_k no later than the time that it enters $v+1$. Consequently, by the prior reasoning, all honest nodes will also receive $\mathcal{C}_{v+1}(B_{k+1})$ and thus will commit B_k. Accordingly, Simple AnonProt commits a new block whenever there are two consecutive honest leaders after GST, which is sufficient to ensure liveness.

IV. Pipelined AnonProt

Although Simple AnonProt has $\omega=\delta$, $\lambda=3\delta$ and reorg resilience, it only provides responsiveness under consecutive honest leaders. If the leader of the current view fails then the next leader has to wait for $\Omega(\Delta)$ time to ensure that it can create a block that will become certified, naturally increasing τ. We now present Pipelined AnonProt (Figure 2), a CRL protocol that improves on Simple AnonProt in both of these areas to achieve full optimistic responsiveness and a τ of 3Δ.

Towards achieving optimistic responsiveness with $\tau=3\Delta$. In Pipelined AnonProt, we separate the fallback case of Simple AnonProt's normal proposal into its own proposal type by enabling L_v to create a *fallback proposal* upon receiving $\mathcal{TC}_{v-1}$. We do so by requiring an honest P_i to include its current lock_i in each $\mathcal{T}$ that it sends. We then allow L_v to propose a block that extends the highest ranked $\mathcal{C}$ included in the $\mathcal{T}_{v-1}$ messages used to construct $\mathcal{TC}_{v-1}$.

While this means that L_v no longer needs to wait $\Omega(\Delta)$ time before proposing in the fallback path—making Pipelined AnonProt optimistically responsive per Definition 6—it also means that L_v may not receive all honest locks before proposing. However, since $\mathcal{TC}$s must be constructed from $2f+1$ timeout messages, the highest ranked block certificate in a given $\mathcal{TC}$ necessarily represents the highest lock among at least $f+1$ honest nodes at the time they sent their timeout messages, which, along with the rules for voting, guarantees that there cannot exist a committable block that was certified for a higher view. This preserves the safety of the protocol. We additionally prevent this modification from breaking liveness by relaxing the voting rule for fallback proposals, allowing P_i to vote for the included block if it directly extends the highest ranked block certificate included in the accompanying $\mathcal{TC}$, even if P_i is locked on a higher ranked block certificate.

Requiring timeout messages to include block certificates naturally increases their size. Similarly, since $\mathcal{TC}$s must provably contain the highest ranked block certificate out of $2f+1$ timeout messages, they are necessarily linear in size even when using threshold signatures [9]. Accordingly, to avoid cubic communication complexity even under threshold signatures, our protocol replaces the $\mathcal{TC}$ multicast of Simple AnonProt with a Bracha-style amplification step [10]. In particular, P_i multicasts a $\mathcal{T}_v$ whilst in view v' where $v' \leq v$ when it first receives either $f+1$ $\mathcal{T}_v$ or $\mathcal{TC}_v$ from other nodes. This ensures that all nodes continue to enter new views after GST: In short, either all honest nodes will send view v Timeout messages, or, since we still require nodes to broadcast block certificates, either some honest node must have observed and broadcasted a view v or higher block certificate, or all honest nodes will send view v'' Timeout messages, where $v'' > v$.

A. Protocol Details

We now present the details of Pipelined AnonProt. We start with refinements to the definition of a block certificate and the certificate ranking rules before elaborating on the steps outlined in Figure 2 that differ from Simple AnonProt.

Block certificates. In Pipelined AnonProt, we use three types of signed vote messages: an optimistic vote (opt-vote), a normal vote (vote) and a fallback vote (fb-vote). Importantly, vote messages with different types may not be aggregated together. Accordingly, we now distinguish between three different types of block certificates. An *optimistic certificate* $\mathcal{C}^o_v(B_h)$ for a block B_h consists of $2f+1$ distinct opt-vote messages for B_h for view v. Similarly, a *normal certificate* $\mathcal{C}^n_v(B_h)$ consists of

A Pipelined AnonProt node P_i runs the following protocol whilst in view v:

1) **Propose.** Upon entering v, the leader L_v proposes using one of the following rules:
   a) **Normal Propose.** If L_v entered v by receiving $\mathcal{C}_{v-1}(B_{k-1})$, multicast $\langle \mathsf{propose}, B_k, \mathcal{C}_{v-1}(B_{k-1}), v\rangle$ where B_k extends B_{k-1}.
   b) **Fallback Propose.** If L_v entered v by receiving $\mathcal{TC}_{v-1}$, multicast a proposal $\langle \mathsf{fb\text{-}propose}, B_k, \mathcal{C}_{v'}(B_{k-1}), \mathcal{TC}_{v-1}, v\rangle$ where $\mathcal{C}_{v'}(B_{k-1})$ is the highest ranked certificate in $\mathcal{TC}_{v-1}$ and B_k extends B_{k-1}.
2) **Vote.** P_i votes at most twice in view v when the following conditions are met:
   a) **Optimistic Vote.** Upon receiving the first optimistic proposal $\langle \mathsf{opt\text{-}propose}, B_k, v\rangle$ where B_k extends B_{k-1}, if (i) $\mathsf{timeout\_view}_i < v-1$, (ii) $\mathsf{lock}_i = \mathcal{C}_{v-1}(B_{k-1})$ and (iii) P_i has not voted in v, multicast an optimistic vote $\langle \mathsf{opt\text{-}vote}, H(B_k), v\rangle_i$ for B_k.
   b) After executing *Advance View* and *Lock* with all embedded certificates, vote once when one of the following conditions are satisfied:
      i) **Normal Vote.** Upon receiving the first normal proposal $\langle \mathsf{propose}, B_k, \mathcal{C}_{v-1}(B_h), v\rangle$, if (i) $\mathsf{timeout\_view}_i < v$, (ii) B_k directly extends B_h and (iii) P_i has not sent an optimistic vote for an equivocating block $B'_{k'}$ in v, multicast $\langle \mathsf{vote}, H(B_k), v\rangle_i$ for B_k.
      ii) **Fallback Vote.** Upon receiving the first fallback proposal $\langle \mathsf{fb\text{-}propose}, B_k, \mathcal{C}_{v'}(B_h), \mathcal{TC}_{v-1}, v\rangle$ if (i) $\mathsf{timeout\_view}_i < v$ and (ii) B_k directly extends B_h and $\mathcal{C}_{v'}(B_h)$ is the highest ranked certificate in $\mathcal{TC}_{v-1}$, multicast $\langle \mathsf{fb\text{-}vote}, H(B_k), v\rangle_i$ for B_k.
3) **Optimistic Propose.** If P_i is L_{v+1} and voted for B_k in view v, multicast $\langle \mathsf{opt\text{-}propose}, B_{k+1}, v+1\rangle$ where B_{k+1} extends B_k.
4) **Timeout.** If $\mathsf{view\text{-}timer}_i$ expires and P_i has not already sent $\mathcal{T}_v$, then multicast $\langle \mathsf{timeout}, v, \mathsf{lock}_i\rangle_i$ and set $\mathsf{timeout\_view}_i = \max(\mathsf{timeout\_view}_i, v)$. Additionally, upon receiving $f+1$ distinct $\langle \mathsf{timeout}, v', \_\rangle_*$ messages or $\mathcal{TC}_{v'}$ such that $v' \geq v$ and not having sent $\mathcal{T}_{v'}$, multicast $\langle \mathsf{timeout}, v', \mathsf{lock}_i\rangle_i$ and set $\mathsf{timeout\_view}_i = \max(\mathsf{timeout\_view}_i, v')$.
5) **Advance View.** P_i enters v' where $v' > v$ using one of the following rules:
   - Upon receiving $\mathcal{C}_{v'-1}(B_h)$. Also, multicast $\mathcal{C}_{v'-1}(B_h)$.
   - Upon receiving $\mathcal{TC}_{v'-1}$. Also, unicast $\mathcal{TC}_{v'-1}$ to $L_{v'}$.

   Finally, reset $\mathsf{view\text{-}timer}_i$ to 3Δ and start counting down.

P_i additionally performs the following actions in any view:

1) **Lock.** Upon receiving $\mathcal{C}_v(B_k)$ whilst having $\mathsf{lock}_i = \mathcal{C}_{v'}(B_{k'})$ such that $v > v'$, set lock_i to $\mathcal{C}_v(B_k)$.
2) **Direct Commit.** Upon receiving $\mathcal{C}_{v-1}(B_{k-1})$ and $\mathcal{C}_v(B_k)$ such that B_k extends B_{k-1}, commit B_{k-1}.
3) **Indirect Commit.** Upon directly committing B_{k-1}, commit all of its uncommitted ancestors.

Fig. 2: The Pipelined AnonProt Protocol

$2f+1$ distinct vote messages for B_h for view v. Finally, a *fallback certificate* $\mathcal{C}^f_v(B_h)$ consists of $2f+1$ distinct fb-vote messages for B_h for view v. We denote a block certificate with $\mathcal{C}_v(B_h)$ whenever its type is not relevant.

Locking. Simple AnonProt only allowed P_i to update lock_i upon entering a new view. In contrast, Pipelined AnonProt requires P_i to update lock_i upon receiving a higher ranked block certificate than its current lock_i, which may happen at any time during the protocol run.

Advance View and Timeout. As in Simple AnonProt, P_i enters view v from some view $v' < v$ upon receiving $\mathcal{C}_{v-1}$ or $\mathcal{TC}_{v-1}$. In the former case, as before, it then multicasts $\mathcal{C}_{v-1}$ to assist with reorg resilience and view synchronization. Comparatively, in the latter case P_i now unicasts $\mathcal{TC}_{v-1}$ to L_v instead of multicasting it. This helps to reduce the communication complexity of the protocol in light of its modified timeout messages, while still ensuring that L_v enters v within Δ of the first honest node doing so after GST. This in turn makes a view-timer of 3Δ sufficient to guarantee the liveness of the protocol, which P_i additionally resets regardless of how it enters v, and starts counting down. As before, if P_i does not advance to a new view before its view timer expires then it multicasts $\langle \mathsf{timeout}, v, \mathsf{lock}_i\rangle_i$. It likewise multicasts the same message for v'' upon observing evidence of at least one honest node requesting a view change for v'' such that $v'' \geq v$. This latter rule differs from Simple AnonProt and compensates for Pipelined AnonProt's removal of $\mathcal{TC}$ multicasting.

Propose. Pipelined AnonProt consists of three distinct ways to propose a new block in a view; i) an optimistic proposal, ii) a normal proposal, and iii) a fallback proposal. An honest node proposes using at most two of the three methods. The optimistic proposal rule remains the same as in Simple AnonProt and serves the same purpose, allowing voting to proceed without delay when network conditions are favourable. Comparatively, the normal proposal rule now only captures the first case of the same rule in Simple AnonProt: Namely, L_v multicasts a normal proposal $\langle \mathsf{propose}, B_k, \mathcal{C}_{v-1}(B_{k-1}), v\rangle$, where B_k extends B_{k-1}, upon entering view v via $\mathcal{C}_{v-1}(B_{k-1})$. As before, L_v does this even if it has already sent an optimistic proposal extending B_{k-1} (which, as before, will necessarily contain B_k). As in Simple AnonProt, this helps Pipelined AnonProt obtain reorg resilience by ensuring that, after GST, L_v will create a proposal that all honest nodes will vote for. Finally, L_v multicasts $\langle \mathsf{fb\text{-}propose}, B_h, \mathcal{C}_{v'}(B_{h-1}), \mathcal{TC}_{v-1}, v\rangle$, where B_h extends B_{h-1} and $\mathcal{C}_{v'}(B_{h-1})$ is the highest ranked $\mathcal{C}$ in $\mathcal{TC}_{v-1}$, upon entering v via $\mathcal{TC}_{v-1}$. We note that in this case, although we have defined the payload of a block as being fixed for a given view, this restriction can be relaxed in the case of fallback proposals, since the voting rules ensure that if $\mathcal{C}^f_v(B_h)$ exists then no other block can be certified for v.

Vote. In Pipelined AnonProt, P_i may vote up to twice in a view; at most once for an optimistic proposal and at most once for either a normal proposal or a fallback proposal. More precisely, P_i multicasts $\langle \mathsf{opt\text{-}vote}, H(B_k), v\rangle_i$ for $\langle \mathsf{opt\text{-}propose}, B_k, v\rangle$, where B_k extends B_{k-1}, when in view v if it has not yet sent a vote for v, or a timeout message for $v-1$ or higher, and has locked $\mathcal{C}_{v-1}(B_{k-1})$. As before, this enables P_i to vote for B_k immediately upon entering v in the best case. Additionally, P_i sends $\langle \mathsf{vote}, H(B_k), v\rangle_i$ for $\langle \mathsf{propose}, B_k, \mathcal{C}_{v-1}(B_{k-1}), v\rangle$ when in v if it has not sent

either an opt-vote for an equivocating block in view v or a timeout message for view v or higher, and B_k extends B_{k-1}. Importantly, P_i must send this vote if it has already sent an optimistic vote for B_k. This ensures that B_k will be certified when L_v is honest and proposes after GST in the case where some honest nodes are unable to send an optimistic vote for B_k. Otherwise, P_i multicasts $\langle \mathsf{fb\text{-}vote}, H(B_h), v\rangle_i$ for $\langle \mathsf{fb\text{-}propose}, B_h, \mathcal{C}_{v'}(B_{h-1}), \mathcal{TC}_{v-1}, v\rangle$ when in view v if it has not sent a timeout message for view v or higher, B_h extends B_{h-1} and $\mathcal{C}_{v'}(B_{h-1})$ is the highest ranked block certificate in $\mathcal{TC}_{v-1}$. Notice that this rule allows P_i to send a fallback vote for B_h after having sent an optimistic vote for an equivocating block, say B_k. However, since the fallback proposal containing B_h can only be valid if it contains $\mathcal{TC}_{v-1}$, at least $f+1$ honest nodes must have sent $\mathcal{T}_{v-1}$ before entering view v and thus will not be able to trigger the optimistic vote rule for B_k, so $\mathcal{C}^o_v(B_k)$ will never exist.

B. Analysis

Why is it safe to vote for a fallback proposal? As we mentioned earlier, we require honest nodes to vote for valid fallback proposals even when they are locked on a higher ranked block certificate than that of the parent of the proposed block. This remains safe because a fallback proposal must be justified by a $\mathcal{TC}$ for the previous view, which in turn contains information about the locks of a majority of the honest nodes. Specifically, $\mathcal{TC}_v$ guarantees that at least $f+1$ nodes had yet to vote for a higher height than $h+1$ upon sending $\mathcal{T}_v$, where h is the height of $\mathcal{C}_{v'}(B_h)$, the highest ranked block certificate included in $\mathcal{TC}_v$. Consequently, there cannot exist a committable block for any height greater than h when $\mathcal{TC}_v$ is constructed. Moreover, if any block can be committed at height h then there can be only one such block. This is because the commit rule only allows a block at height h proposed for view v'' to be committed if its child becomes certified in $v''+1$. Therefore, if B_h can be committed then at least $f+1$ honest nodes must have voted for its child in $v'+1$, and since an honest node cannot vote for a block unless it possesses the block certificate for its parent, these nodes must have had $\mathcal{C}_{v'}(B_h)$ when they did so. Consequently, every $\mathcal{TC}$ for $v'+1$ or higher will necessarily contain $\mathcal{C}_{v'}(B_h)$ or a block certificate for one of its descendants as its highest ranked block certificate, meaning that every fallback proposal for $v'+1$ or higher will necessarily extend B_h. Moreover, by extension, so will every subsequent optimistic or normal proposal.

V. Commit Moonshot

Until now, we have measured λ in terms of δ, however, this is imprecise because δ provides no way of differentiating between the performance of protocols that exchange one type of message for another. Specifically, the pipelining technique replaces two consecutive rounds of voting for one block proposal, with one round of voting for two consecutive block proposals. This means that the commit latency of a block in the pipelined setting is proportional to the dissemination time of not only the block itself, but also its child (in the best case).

Commit AnonProt can be obtained by augmenting the protocol presented in Figure 2 with following steps:

1) **Pre-commit.** Upon receiving $\mathcal{C}_v(B_k)$ whilst in view v, if $\mathsf{timeout\_view}_i < v$, multicast $\langle \mathsf{commit}, H(B_k), v\rangle_i$.
2) **Direct Commit.** Upon receiving a quorum of distinct $\langle \mathsf{commit}, H(B_k), v\rangle_*$ whilst in any view, commit B_k.

Fig. 3: Commit AnonProt

More to the point, pipelining essentially exchanges the cost of disseminating a second vote for the cost of disseminating a second proposal, effectively reducing commit latency when proposals take longer to disseminate than blocks.

We characterize this behavior using the *modified partially synchronous model* of Blum et al. [8] and assume that small messages (in this case, votes) are delivered within ρ (δ in the original formulation) time while large messages (in this case, block proposals) are delivered within Λ time after GST. Under this model, Simple AnonProt and Pipelined AnonProt both incur $\lambda = 2\Lambda + \rho$.

We now present a protocol with $\lambda = \Lambda + 2\rho$, which we call Commit AnonProt. Accordingly, when $\rho < \Lambda$ (which we assume is typically the case in practice), this protocol provides improved commit latency over those presented previously. Commit AnonProt obtains this result through the use of explicit commit messages. Although this modification is theoretically compatible with pipelining, we present it in the non-pipelined setting to obtain our result for Theorem 2. Commit AnonProt also provides $\omega = \delta$ (Λ), reorg resilience and optimistic responsiveness. Moreover, while Simple AnonProt and Pipelined AnonProt require two consecutive honest leaders to be guaranteed to commit a block after GST, Commit AnonProt requires only one.

We present the two modifications required to convert Figure 2 to Commit AnonProt in Figure 3. We briefly discuss the intuition for the safety of these modifications below.

Safety intuition. Per the updated commit rule given in Figure 3, P_i commits B_k and all of its uncommitted ancestors upon receiving a quorum (i.e. $2f+1$ when $n = 3f+1$) of distinct $\langle \mathsf{commit}, H(B_k), v\rangle_*$ messages. This remains safe because $2f+1$ such messages can only exist if at least $f+1$ honest nodes do not send $\mathcal{T}_v$. Consequently, if any block becomes certified for $v+1$ then it must have been proposed in either an optimistic or normal proposal and thus must be a child of B_k. Otherwise, $\mathcal{TC}_{v+1}$ will contain $\mathcal{C}_v(B_k)$ as its highest ranked block certificate and therefore every subsequently certified block will necessarily extend B_k.

VI. Implementation and Evaluation

As shown in Table I, Pipelined AnonProt and Commit AnonProt equal or surpass the theoretical performance of all prior quadratic CRL protocols in all considered metrics. The primary question that remains, then, is whether their increased communication complexity relative to linear protocols is justified. Accordingly, we decided to implement our protocols and evaluate them against Jolteon, a linear protocol with state-of-the-art performance in most metrics and several high-quality

TABLE II: Performance vs Jolteon ($f' = 0$, Outliers Removed)

	Throughput Increase (%)				Latency Reduction (%)			
Prot.	**Max**	$\bar{x}$	$\tilde{x}$	**Min**	**Max**	$\bar{x}$	$\tilde{x}$	**Min**
SA	72	53	55	33	56	43	42	37
PA	70	51	54	24	56	43	42	32
CA	74	52	54	25	69	54	58	38

open-source implementations, which we modified for our own implementations ([5] provides more details).

Our goal was to compare the throughput and latency of our AnonProt protocols with that of Jolteon under varying conditions. Firstly, we would evaluate the trade-off between λ, ω and steady-state communication complexity by running both protocols with $f' = 0$, where f' denotes the number of actual failures in the system (i.e. $f' \leq f = \frac{n-1}{3}$), under varying network and payload sizes. Secondly, we would evaluate the impact of τ and reorg resilience by running each protocol in a single network with $f' = f$ and varying leader schedules. In doing these experiments we also naturally evaluated our protocols against each other, allowing us to illuminate the trade-offs incurred by pipelining and optimistic responsiveness.

We established two metrics for throughput: Firstly, the number of blocks committed by at least $2f + 1$ nodes in the network during a run, hereafter referred to as *throughput*; and secondly, the average number of bytes of payload data transferred per second during the run, hereafter referred to as *transfer rate*. Correspondingly, for latency, we measured the average time between the creation of a block and its commit by the $2f + 1th$ node.

Our networks were constructed from m5.large AWS EC2 instances running Ubuntu 20.04. Each instance had a network bandwidth of up to 10Gbps[3], 8GB of memory and Intel Xeon Platinum 8000 series processors with 2 virtual cores. The instances were evenly distributed across the us-east-1 (N. Virginia), us-west-1 (N. California), eu-north-1 (Stockholm), ap-northeast-1 (Tokyo) and ap-southeast-2 (Sydney) regions.

We refer to Simple AnonProt, Pipelined AnonProt, Commit AnonProt and Jolteon as SA, PA, CA and J in the accompanying figures.

A. Happy Path Evaluation

We first configured all nodes to be honest in order to evaluate the trade-off between λ, ω and steady-state communication complexity. We initially tested networks of 10, 50, 100 and 200 nodes with payload sizes ranging from empty to 1.8MB to understand how the tested protocols scale under these variables. We subsequently tested additional payload sizes in the 200 node network to discover the approximate maximum transfer rate of each protocol in this setting. We ran each configuration three times, with each run lasting five minutes, and averaged the results.

[3]https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-network-bandwidth.html

As shown in Figure 4 and Table II, all AnonProt protocols produced significantly improved throughput and latency compared to Jolteon in all tested configurations. The 200 node network produced significant outliers under the empty and 1.8kB payload configurations, with all three protocols exhibiting $214\% - 230\%$ higher throughput and $70.5\% - 71.8\%$ lower latency than Jolteon, compared to the respective maximums of $70.3\% - 74.4\%$ and $55.7\% - 69.0\%$ seen across all other configurations. Simple AnonProt and Pipelined AnonProt produced near-identical performance in both metrics for most configurations due to the similarity of their happy-path protocols. Conversely, although Commit AnonProt produced similar throughput to these protocols, it exhibited substantially lower latency for payloads above 18kB due to its explicit commit messages, clearly showing the inefficiency of pipelining when blocks are large. Generally speaking, all three AnonProt protocols produced increasingly higher throughput and relatively consistent improvements to latency compared to Jolteon as the network size increased, showing that linear communication complexity is not worthwhile if it comes at the cost of reduced ω amd λ. Finally, per Figure 6, all three protocols achieved a higher maximum transfer rate with lower latency than Jolteon in the 200 node network, with Commit AnonProt producing the best results. Overall, these results show that the happy paths of our AnonProt protocols scale well and provide meaningfully decreased latency and increased throughput compared to Jolteon under the experimental conditions, with Commit AnonProt being the most efficient option.

B. Fallback Path Evaluation

We subsequently evaluated the impact of τ, reorg resilience, optimistic responsiveness and pipelining by running all protocols with a fixed payload size and $f' = f$. We used a fixed network size due to all protocols exhibiting $O(n^2)$ communication complexity in the fallback path, meaning that varying n would provide little new information over the previously-presented results (with one notable exception discussed later). We also used three different fair LSO/LCO leader schedules. The first ($\mathcal{B}$) had all honest nodes followed by all byzantine nodes, representing the best case for non-reorg-resilient and pipelined protocols. The second ($\mathcal{WM}$) had honest-then-byzantine leaders for $2f'$ views, followed by honest leaders for the remaining $n - 2f'$ views, representing the worst case for reorg resilient, pipelined protocols. The third ($\mathcal{WJ}$) repeated two-honest-then-byzantine for $3f'$ views, followed by the remaining $n - 3f'$ honest, representing the worst case for non-reorg resilient, non-pipelined protocols.

We chose $n = 50$, $f = 16$, $\Delta = 1s$ and empty blocks. This configuration maximised the impact of the quadratic steady-state complexity of our protocols while still ensuring that each protocol would make it through several iterations of the leader schedules within the five minute duration of each run.

As shown in Figures 7 and 8, Jolteon's performance degrades enormously in the presence of failures due to its lack of reorg resilience. This is evident by the difference in its results for $\mathcal{B}$ and $\mathcal{WJ}$, with the former producing 180%

Fig. 4: Performance Overview ($f' = 0$, $p \leq 1.8$MB)

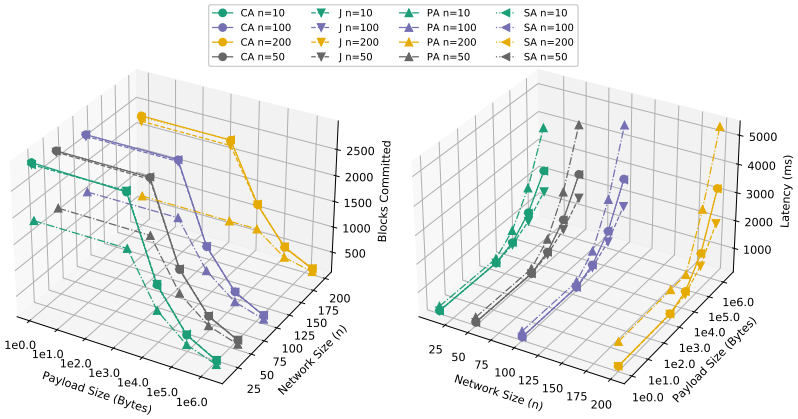


Fig. 5: Performance vs. Jolteon ($f' = 0$, No Outliers)

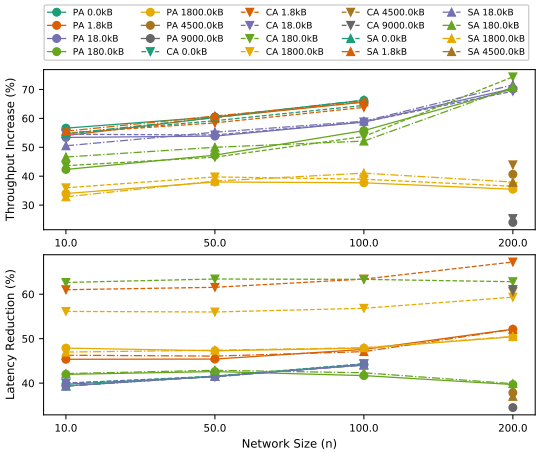
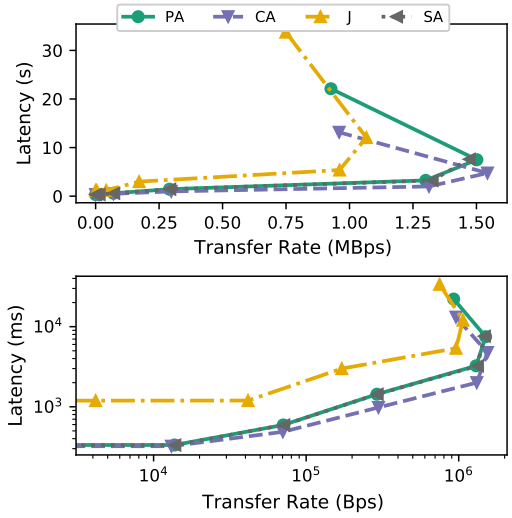


Fig. 6: Throughput vs Latency ($n = 200$, $f' = 0$, $p \leq 9$MB)



higher throughput and 96.6% lower latency than the latter. The pipelined nature of Simple AnonProt and Pipelined AnonProt likewise caused a significant reduction in latency between the worst ($\mathcal{WM}$) and best case ($\mathcal{B}$) leader schedules for these protocols. Simple AnonProt's 2Δ wait after a failed leader (i.e. lack of Optimistic Responsiveness) caused its performance to vary more significantly than Pipelined AnonProt, while its longer view length caused a substantial decrease in throughput.

As shown by their absence from Figure 9, both Simple AnonProt and Pipelined AnonProt failed to improve over Jolteon under $\mathcal{WM}$. More precisely, although they respectively produced 91% and 383% higher throughput than Jolteon, their latencies were also 208% and 542% higher. Both of these results were a side-effect of their reorg resilience: Both AnonProt protocols committed all blocks proposed by honest leaders with Byzantine successors under this schedule, but only after a significant delay. Comparatively, Jolteon lost all such blocks due to lacking this property, with only the final block of the final honest leader in the schedule being committed with a delay. Accordingly, Jolteon's relative improvement in block commit latency over these two protocols should increase proportionally to n, while its relative throughput should similarly decrease. We note that in this case decreased block commit latency at the cost of decreased throughput should be considered an undesirable trade-off as it does not imply a decrease in transaction commit latency.

Finally, Commit AnonProt performed consistently well regardless of the leader election schedule due to its lack of pipelining, which denies the adversary any power to delay the commit of honest blocks. Notably, it produced 417% higher throughput and 99.3% (151x) lower latency than Jolteon under $\mathcal{WJ}$. Overall, then, Commit AnonProt produced superior performance in both the happy path and the fallback path.

VII. Related Work

There has been a long line of work towards designing efficient BFT SMR protocols for partially synchronous networks (which we cite further on). Our work contributes to this effort by introducing the first CRL protocols to obtain both $\omega = \delta$ and $\lambda = 3\delta$. Our protocols further provide reorg resilience, improving their recovery time after a failed leader compared to prior chain-based works that fail to achieve this property. This is especially true of both Pipelined AnonProt and Commit AnonProt, which also have low τ and are optimistically responsive. These properties come at the cost of $O(n^2)$ steady-state communication complexity, making our protocols less performant in this metric compared to vote-aggregator-based protocols like HotStuff, however, as shown in Section VI, this trade-off is worthwhile in many settings. We presented a brief comparison between our protocols and other recent works in Section I. We now undertake a more thorough review.

Early works. PBFT [13] was the first practical BFT SMR protocol, achieving $\lambda = 3\delta$ at the cost of $O(n^2)$ steady-state communication. PBFT's slot-based nature complicated its view change, leading it to only rotate leaders after a failure—an approach that allows proposal frequency to be reduced below δ, but precludes fairness. Much later, Tendermint [11] combined the steady-state and view-change sub-protocols into

Fig. 7: Performance Overview ($n = 50$, $f' = 16$, $p = 0$)

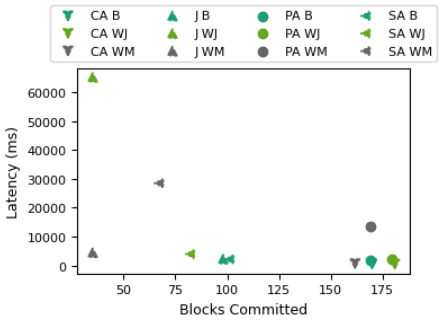


Fig. 8: Performance Overview (Log Scaled) ($n = 50$, $f' = 16$, $p = 0$)

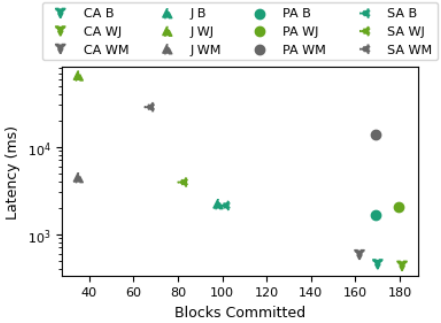
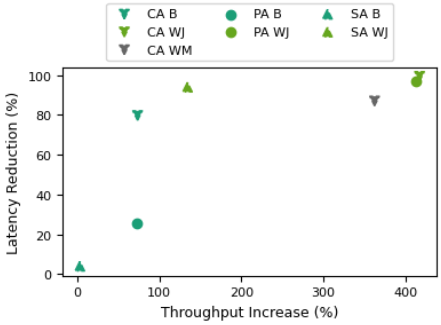


Fig. 9: Improvements vs. Jolteon ($n = 50$, $f' = 16$, $p = 0$)



a unified protocol for the LSO setting, resulting in a simpler protocol than PBFT at the cost of an $\Omega(\Delta)$ wait before every new view at the same height, thus sacrificing optimistic responsiveness. HotStuff [33] formalized the notion of optimistic responsiveness and improved upon Tendermint both by implementing this property and being the first protocol to obtain linear ($O(n)$) communication complexity in both its steady-state and view-change phases (in the presence of an abstract pacemaker for view synchronization). To our knowledge, it was also the first protocol to implement block chaining.

Linear protocols. Like HotStuff, many other chain-based protocols [20], [21], [18], [32], [25] have focused on minimising communication complexity, with some achieving linearity only in their steady states and others during their view-change phases as well. Recently, some [16], [25] have even achieved amortized-linear view synchronization. In all cases though, these protocols obtain steady-state linearity through the use of a designated vote-aggregator node. As we previously observed, this naturally increases their λ, ω and τ relative to our protocols, and precludes reorg resilience when the aggregator is not the original proposer. Moreover, while most nodes incur a steady-state complexity of $O(1)$ in these protocols, the proposer must still send and the aggregator must still receive, $O(n)$ messages. This imbalance means that these protocols under-utilize the available bandwidth in the point-to-point CRL setting (in which there should be no choke-points in the network and each node should have similar capabilities).

Non-linear pipelined chain-based protocols. PaLa [15] is a pipelined CRL protocol with $\lambda = 4\delta$ and $\omega = 2\delta$. While this improves upon the commit latencies of linear pipelined protocols with $\omega = 2\delta$, like [18], PaLa achieves this result at the cost of $O(n^2)$ communication complexity in its steady state. Accordingly, PaLa is sub-optimal in all three properties.

Non-pipelined chain-based protocols. Similar to PaLa, ICC [12] incurs $O(n^2)$ steady-state communication complexity. However, this protocol eschews pipelining, allowing it to achieve $\lambda = 3\delta$ through the use of an explicit second round of voting for each block. Even so, it lacks reorg resilience and its ω of 2δ and τ of 4Δ make it less efficient in these metrics than our protocols. Simplex [14] obtains the same λ and ω with $\tau = 3\Delta$, however, it claims responsiveness only when all nodes are honest. Additionally, its requirement that a leader must send the entire certified blockchain along with its proposal makes its communication complexity proportional to size of the blockchain and thus unbounded, rendering it impractical.

Apollo [6]. To the best of our knowledge, Apollo is the only chain-based protocol with $\omega = \delta$. However, it has $\lambda = (f+1)\delta$ even during failure-free executions and assumes a synchronous communication model.

DAG-based protocols. DAG-based consensus protocols like [30], [31], [26], [22] focus on improving block throughput. While they naturally produce and commit more blocks over a given interval than chain-based protocols by virtue of having all nodes propose in each step, they incur $O(n^3)$ communication in doing so. While recent protocols [22], [26] in this setting have achieved $\omega = \delta$, and $\lambda = 3\delta$ for blocks proposed by the leader, they require at least 4δ to commit blocks proposed by other nodes. Consequently, since most blocks committed by these protocols are non-leader blocks, their average commit latency is still higher than our protocols. Moreover, since these protocols use pipelining, each δ corresponds to one Λ under our model from Section V, meaning that these latencies become even more significant relative to our protocols as block size increases.

Inspiration for future work. A related line of works [1], [23], [27], [3], [20] has explored optimistic commits with a latency of 2δ when all nodes behave honestly or in the presence $5f - 1$ or more nodes in the system. Additionally, very recently, Giridharan et al. [19] proposed BeeGees, a new pipelined chained BFT SMR protocol that is able to commits without requiring consecutive honest leaders. Investigating the integration of these optimizations into our protocols represents an interesting direction for future work.

References

[1] Michael Abd-El-Malek, Gregory R Ganger, Garth R Goodson, Michael K Reiter, and Jay J Wylie. Fault-scalable byzantine fault-tolerant services. *ACM SIGOPS Operating Systems Review*, 39(5):59–74, 2005.

[2] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: Simple and practical synchronous state machine replication. In *IEEE S&P*, pages 106–118. IEEE, 2020.

[3] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of byzantine broadcast: A complete categorization. In *PODC*, pages 331–341, 2021.

[4] Ittai Abraham, Kartik Nayak, and Nibesh Shrestha. Optimal good-case latency for rotating leader synchronous bft. In *OPODIS*, 2022.

[5] Anonymous Author(s). Anonprot full version. https://anonymous.4open.science/r/anonprot-C64F/anonprot.pdf.

[6] Adithya Bhat, Akhil Bandarupalli, Saurabh Bagchi, Aniket Kate, and Michael Reiter. Unique chain rule and its applications. In *FC*, 2023.

[7] bitfly gmbh. Validators chart. https://beaconcha.in/charts/validators. [Online; accessed 04-December-2023].

[8] Erica Blum, Derek Leung, Julian Loss, Jonathan Katz, and Tal Rabin. Analyzing the real-world security of the algorand blockchain. In *CCS*, pages 830–844, 2023.

[9] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *Journal of cryptology*, 17:297–319, 2004.

[10] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[11] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, University of Guelph, 2016.

[12] Jan Camenisch, Manu Drijvers, Timo Hanke, Yvonne-Anne Pignolet, Victor Shoup, and Dominic Williams. Internet computer consensus. In *PODC*, pages 81–91, 2022.

[13] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[14] Benjamin Y Chan and Rafael Pass. Simplex consensus: A simple and fast consensus protocol. In *TCC 2023 (To appear)*. Springer, 2023.

[15] TH Hubert Chan, Rafael Pass, and Elaine Shi. Pala: A simple partially synchronous blockchain. *Cryptology ePrint Archive*, 2018.

[16] Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. Byzantine consensus is θ (n^2): The dolev-reischuk bound is tight even in partial synchrony! In *DISC*, 2022.

[17] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[18] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In *FC*, pages 296–315, 2022.

[19] Neil Giridharan, Florian Suri-Payer, Matthew Ding, Heidi Howard, Ittai Abraham, and Natacha Crooks. Beegees: stayin'alive in chained bft. In *PODC*, pages 233–243, 2023.

[20] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: A scalable and decentralized trust infrastructure. In *DSN*, pages 568–580. IEEE, 2019.

[21] Mohammad M Jalalzai, Jianyu Niu, Chen Feng, and Fangyu Gai. Fast-hotstuff: A fast and resilient hotstuff protocol. *arXiv preprint arXiv:2010.11454*, 2020.

[22] Idit Keidar, Oded Naor, Ouri Poupko, and Ehud Shapiro. Cordial miners: Fast and efficient consensus for every eventuality. In *DISC*, 2023.

[23] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative byzantine fault tolerance. In *SOSP*, pages 45–58, 2007.

[24] Aptos Labs. Validators. https://explorer.aptoslabs.com/validators/all?network=mainnet. [Online; accessed 04-December-2023].

[25] Dahlia Malkhi and Kartik Nayak. Hotstuff-2: Optimal two-phase responsive bft. *Cryptology ePrint Archive*, 2023.

[26] Dahlia Malkhi, Chrysoula Stathakopoulou, and Maofan Yin. Bbca-chain: One-message, low latency bft consensus on a dag. *arXiv preprint arXiv:2310.06335*, 2023.

[27] J-P Martin and Lorenzo Alvisi. Fast byzantine consensus. *TDSC*, 3(3):202–215, 2006.

[28] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *DISC*, page 6, 2017.

[29] Elaine Shi. Streamlined blockchains: A simple and elegant approach (a tutorial and survey). In *ASIACRYPT*, pages 3–17. Springer, 2019.

[30] Alexander Spiegelman, Balaji Aurn, Rati Gelashvili, and Zekun Li. Shoal: Improving dag-bft latency and robustness. *arXiv preprint arXiv:2306.03058*, 2023.

[31] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: Dag bft protocols made practical. In *CCS*, pages 2705–2718, 2022.

[32] Xiao Sui, Sisi Duan, and Haibin Zhang. Marlin: Two-phase bft with linearity. In *DSN*, pages 54–66. IEEE, 2022.

[33] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *PODC*, pages 347–356, 2019.

APPENDIX A
SIMPLE ANONPROT SECURITY ANALYSIS

We now present formal proofs that Simple AnonProt satisfies the safety and liveness properties of SMR, and reorg resilience.

Claim 1 (Quorum Intersection)**.** *Given any two quorums Q_1 and Q_2 drawn from $\mathcal{V}$, Q_1 and Q_2 have at least one honest node in common.*

Proof. According to the definition given in Section II, when $n = 3f + 1$ a quorum contains $2f + 1$ distinct members of $\mathcal{V}$. Therefore, Q_1 and Q_2 must have at least $f + 1$ members in common. Thus, because $\mathcal{V}$ contains only f Byzantine nodes, at least one of these shared nodes must be honest. □

Claim 2 (Honest Majority Intersection)**.** *Given any two sets H_1 and H_2 of at least $f + 1$ honest nodes drawn from $\mathcal{V}$, H_1 and H_2 have at least one honest node in common.*

Proof. According to the definition given in Section II, when $n = 3f + 1$, $\mathcal{V}$ contains $2f + 1$ honest nodes. Therefore, since H_1 and H_2 both contain at least $f+1$ honest nodes, they must have at least one node in common. □

Lemma 1. *If $\mathcal{C}_v(B_k)$ and $\mathcal{C}_v(B_l)$ exist then $B_k = B_l$.*

Proof. Suppose, for the sake of contradiction, that $\mathcal{C}_v(B_k)$ and $\mathcal{C}_v(B_l)$ exist but $B_k \neq B_l$. By the definition of a block certificate given in Section II, the existence of $\mathcal{C}_v(B)$ implies that at least $2f + 1$ nodes voted for B_k in view v. Likewise, the existence of $\mathcal{C}_v(B_l)$ implies that the same number of nodes also voted for B_l in v. By Lemma 1, this implies that at least one honest node voted for both B_k and B_l in v. However, this violates the rules for voting, which allow a node to vote only once per view, contradicting the original assumption. □

Claim 3. *If an honest node votes for $\langle \mathsf{propose}, B_l, \mathcal{C}_{v'}(B_h), v\rangle$, then $v' < v$.*

Proof. Since the view advancement rule takes priority over the voting rule, if $v' \geq v$ then an honest node would have entered $v'+1 > v$ before voting for $\langle \mathsf{propose}, B_l, \mathcal{C}_{v'}(B_h), v\rangle$, making it ineligible to vote for this proposal. □

Lemma 2. *If an honest node, say P_i, directly commits a block B_k that was certified for view v and $\mathcal{C}_{v'}(B_{k'})$ exists such that $v' = v$ or $v' = v + 1$, then $B_{k'}$ extends B_k.*

Proof. If $v' = v$ then, by Lemma 1, $B_{k'} = B_k$ and thus, per the definition of block extension given in Section II, $B_{k'}$ extends B_k. Alternatively, if $v' = v+1$ then, by the direct commit rule, P_i must have observed $\mathcal{C}_v(B_k)$ and $\mathcal{C}_{v+1}(B_{k+1})$ with B_{k+1} extending B_k. Additionally, by Lemma 1, $\mathcal{C}_{v+1}(B_{k+1})$ is the only block certificate that can exist for $v + 1$. Thus, $B_{k'} = B_{k+1}$, so $B_{k'}$ extends B_k. □

Lemma 3 (Unique Extensibility)**.** *If an honest node, say P_i, directly commits a block B_k that was certified for view v and $\mathcal{C}_{v'}(B_{k'})$ exists such that $v' \geq v$, then $B_{k'}$ extends B_k.*

Proof. We complete this proof by strong induction on v', however, Lemma 2 covers the base cases ($v' = v$ and $v' = v + 1$) so we proceed directly with the inductive step.

Inductive step: $v' > v + 1$. For our induction hypothesis, we assume that the lemma holds up to view $v' - 1$. That is, we assume that every $\mathcal{C}_{v^*}(B_{k^*})$ with $v \leq v^* < v'$ extends B_k. We use this assumption to prove that it also holds for v'. We first observe that the existence of $\mathcal{C}_{v'}(B_{k'})$ implies that a set H_1 of at least $f + 1$ honest nodes voted for $B_{k'}$ in view v'. If any of these nodes voted using the optimistic vote rule, then they must have been locked on $\mathcal{C}_{v'-1}(B_{k'-1})$ and $B_{k'}$ must extend $B_{k'-1}$. Therefore, since by the induction hypothesis $B_{k'-1}$ extends B_k, $B_{k'}$ also extends B_k. Alternatively, if no honest node used the optimistic vote rule to vote for $B_{k'}$ then all members of H_1 must have used the normal vote rule to vote for $B_{k'}$. Therefore, they must have received $\langle \mathsf{propose}, B_{k'}, \mathcal{C}_{v''}(B_{k''}), v'\rangle$ such that $B_{k'}$ extended $B_{k''}$ and $\mathcal{C}_{v''}(B_{k''})$ ranked equal to or higher than their respective locks. Moreover, by Claim 3, $v'' < v'$. We now show that $v'' \geq v$.

Recall that we know from the commit rule that $\mathcal{C}_{v+1}(B_{k+1})$ exists and that B_{k+1} extends B_k. Therefore, a set, say H_2, of at least $f+1$ honest nodes must have voted for B_{k+1} in view $v + 1$. Furthermore, by the vote rules, they must have done this after receiving $\mathcal{C}_v(B_k)$ and therefore would have locked $\mathcal{C}_v(B_k)$ or a higher ranked block certificate upon advancing to a new view. By Claim 2, H_1 and H_2 must have at least one member, say P_i, in common. Since the view advancement rule ensures that P_i never decreases its local view, it must have voted for $B_{k'}$ (in v') after B_{k+1} (in $v+1$) and thus must have been locked on $\mathcal{C}_v(B_k)$ or higher upon doing so. Hence, since the normal vote rule ensures that $\mathcal{C}_{v''}(B_{k''}) \geq \mathsf{lock}_i$ for P_i, by the block certificate ranking rule, $v'' \geq v$. Hence, since $v \leq v'' < v'$, by the induction hypothesis, $B_{k''}$ extends B_k, so $B_{k'}$ also extends B_k. □

Theorem 3 (Safety)**.** *Honest nodes do not commit different values at the same log position.*

Proof. We show that if two honest nodes P_i and P_j commit B_k and B'_k, then $B_k = B'_k$. This fact together with the assumptions mentioned along with Definition 1 in Section II, is sufficient to achieve safety.

Suppose, for the sake of contradiction, that P_i and P_j commit B_k and B'_k but $B_k \neq B'_k$. By the indirect commit rule, P_i and P_j must do so as a result of respectively directly committing blocks B_l and B_m such that B_l extends B_k and $l \geq k$, and B_m extends B'_k and $m \geq k$. Thus, by Lemma 3, either $v \leq v'$ and B_m extends B_l, or $v \geq v'$ and B_l extends B_m. Therefore, since B_l and B_m are a part of the same chain and because each block in the chain has exactly one parent, $B_k = B'_k$. □

Claim 4. *Let t_g denote GST. If the first honest node to enter view v does so at time t, then all honest nodes enter v or higher by $\max(t_g, t) + \Delta$.*

Proof. Let P_i be the first honest node to enter v. By the view advancement rule, it must have entered v via either $\mathcal{C}_{v-1}$ or

$\mathcal{TC}_{v-1}$ and must have multicasted this certificate upon doing so. Therefore, since messages sent by honest nodes arrive within Δ time after GST, all honest nodes will receive this certificate by $\max(t_g, t) + \Delta$ and thus will enter v if they have not already entered a higher view. □

Lemma 4. *All honest nodes keep entering increasing views.*

Proof. Suppose, for the sake of contradiction, that at least one honest node, say P_i, becomes stuck in view v and let v' be the highest view of any honest node at any time. If $v' > v$ then Claim 4 shows that P_i will enter v' or higher, contradicting the assumption that it becomes stuck in v. Otherwise, if $v' = v$ then since this implies that no honest node ever enters a view higher than v and because Claim 4 shows that all honest nodes will enter v, they must all become stuck there. However, by the view advancement and timeout rules, these nodes will all eventually multicast $\mathcal{T}_v$ and thus will all be able to construct $\mathcal{TC}_v$ and enter $v+1$, contradicting the conclusion that they must become stuck in v. □

Claim 5. *If an honest node enters view v then at least $f+1$ honest nodes must have already entered $v-1$.*

Proof. The view advancement rule requires an honest node to observe either $\mathcal{C}_{v-1}$ or $\mathcal{TC}_{v-1}$ in order to enter v. Therefore, at least $f+1$ honest nodes must send the corresponding messages. Moreover, by the vote and timeout rules, they must do so whilst in v. Thus, in either case, an honest node can only enter v if at least $f+1$ honest nodes have already entered $v-1$. □

Lemma 5. *If the first honest node to enter view v does so after GST and L_v is honest, then all honest nodes receive $\mathcal{C}_v(B_k)$ for some block B_k proposed by L_v, and at least $f+1$ of them lock this certificate while entering $v+1$.*

Proof. By Lemma 1, only one block can become certified for a given view. Thus, if $\mathcal{C}_v(B_k)$ exists then any node that receives a view v block certificate must receive $\mathcal{C}_v(B_k)$. We assume this fact in the remainder of the proof.

Let t be the time when the first honest node enters view v. Because honest nodes only send $\mathcal{T}_v$ either after receiving $f+1$ such messages from unique senders, or upon their view timers expiring, no honest node will send $\mathcal{T}_v$ until $t+5\Delta$. Moreover, since by Claim 5 no honest node can enter a view greater than v until at least $f+1$ honest nodes enter v, neither can any honest node send a timeout message for a view greater than v before this time. Thus, no honest node can enter a view greater than v via a timeout certificate before $t+5\Delta$. Consequently, since by the same lemma no honest node can enter a view greater than $v+1$ unless at least $f+1$ honest nodes first enter $v+1$, if any honest node enters a view greater than $v+1$ before $t+5\Delta$ then, by the view advancement rule, at least $f+1$ honest nodes must have locked and multicasted $\mathcal{C}_v(B_k)$ upon entering $v+1$, so the proof is complete. Furthermore, if any honest node enters $v+1$ before $t+4\Delta$ then, by the view advancement rule, it will multicast $\mathcal{C}_v$ upon doing so, which all nodes will receive before $t+5\Delta$. Therefore, either all honest nodes enter $v+1$ and lock $\mathcal{C}_v$ before $t+5\Delta$, or some honest node enters a view greater than $v+1$ before $t+5\Delta$. In either case, the proof is complete.

Suppose, then, both that no honest node enters a view greater than $v+1$ before $t+5\Delta$ and that no honest node enters $v+1$ before $t+4\Delta$. Therefore, by Claim 4, all honest nodes will enter v before $t+\Delta$. If L_v enters v via $\mathcal{C}_{v-1}(B_h)$, then it will multicast $\langle \mathsf{propose}, B_{h+1}, \mathcal{C}_{v-1}(B_h), v\rangle$ with B_{h+1} extending B_h, which all honest nodes will receive before $t+2\Delta$. Therefore, if all honest nodes vote for B_{h+1} no later than the time that they receive this proposal then they will all receive $\mathcal{C}_v(B_{h+1})$ before $t+3\Delta$. Thus, by Claim 5, at least $f+1$ of them will enter $v+1$ via this certificate and will subsequently lock it, completing the proof. Otherwise, some honest node, say P_j, must fail to vote for B_{h+1} before $t+2\Delta$. However, since we have already considered the case where any honest node enters a view greater than v before $t+4\Delta$, P_j cannot have $\mathsf{lock}_i > \mathcal{C}_{v-1}(B_h)$ when it attempts to vote for B_{h+1}. Therefore, since L_v will ensure that B_{h+1} extends B_h, P_j can only have failed to vote for B_{h+1} if it had already voted in view v. However, since L_v is honest it will only create a single normal proposal, so P_j must have voted for an optimistic proposal containing some block B_k. However, since Lemma 1 shows that only one block can become certified for $v-1$, by the optimistic vote rule, B_k extends B_h. Moreover, since we have defined block payloads as being fixed for a given view, because L_v is honest, B_k must also have the same payload as B_{h+1}. Thus, $B_k = B_{h+1}$, contradicting the conclusion that P_j must have failed to vote for B_k and completing the proof.

Otherwise, if L_v enters v via $\mathcal{TC}_{v-1}$ then it will wait 2Δ before proposing. As before, this implies that all honest nodes enter v before $t+\Delta$. By the view advancement rule, any node that does so via $\mathcal{TC}_{v-1}$ will unicast $\langle \mathsf{status}, v, \mathsf{lock}_i\rangle$ to L_v. Similarly, any node that enters v via $\mathcal{C}_{v-1}$ will multicast this certificate. Consequently, L_v will receive the highest ranked block certificate, say $\mathcal{C}_{v'}(B_h)$, known to any honest node before $t+3\Delta$. Thus, since L_v is honest, when it proposes it will multicast a normal proposal containing a block that extends B_h; i.e., $\langle \mathsf{propose}, B_{h+1}, \mathcal{C}_{v'}(B_h), v\rangle$. All honest nodes will receive this proposal before $t+4\Delta$. Furthermore, if they all vote for B_{h+1} before this time then they will all receive $\mathcal{C}_v(B_{h+1})$ before $t+5\Delta$. Thus, since we have already concluded that no honest node can enter $v+1$ or higher via a timeout certificate before this time, by Claim 5, at least $f+1$ of them will lock $\mathcal{C}_v(B_{h+1})$ upon entering $v+1$. Otherwise, some honest node, say P_j, must fail to vote for B_{h+1} before $t+4\Delta$. However, we already know that all honest nodes will enter v before $t+\Delta$ and will have $\mathsf{lock}_i \leq \mathcal{C}_{v'}(B_h)$ upon receiving L_v's proposal, which will occur before $t+4\Delta$ and thus before any of them can have sent $\mathcal{T}_v$. Moreover, as previously reasoned, L_v will not create an equivocal proposal that P_j can vote for. Therefore, P_j must vote for B_{h+1} before $t+4\Delta$. Thus, as before, all honest nodes will receive $\mathcal{C}_v(B_{h+1})$ before $t+5\Delta$ and since no honest node can enter $v+1$ or higher via a timeout certificate before this time, by Claim 5,

at least $f+1$ of them will lock this certificate upon entering $v+1$, completing the proof. □

Lemma 6. *If the first honest node to enter view v does so after GST, L_v is honest and proposes a block B_k that becomes certified, and $\mathcal{C}_{v+1}(B_l)$ exists, then B_l directly extends B_k.*

Proof. By Lemma 5, a set H_1 of at least $f+1$ honest nodes lock $\mathcal{C}_v(B_k)$ while entering $v+1$. Furthermore, $\mathcal{C}_{v+1}(B_l)$ can only exist if a set H_2 of at least $f+1$ honest nodes vote for B_l in view $v+1$. By Claim 2, H_1 and H_2 must have at least one node, say P_i, in common. Thus, since the optimistic vote rule requires P_i to be locked on the parent of B_l, if P_i votes for an optimistic proposal containing B_l then B_l must directly extend B_k. Alternatively, P_i must vote for $\langle \mathsf{propose}, B_l, \mathcal{C}_{v'}(B_h), v+1\rangle$. Thus, since by Lemma 1 and Claim 3 $\mathcal{C}_{v'}(B_h) = \mathcal{C}_v(B_k)$, B_l must directly extend B_k. □

Theorem 4 (Liveness)**.** *Each client request is eventually committed by all honest nodes.*

Proof. We show that all honest nodes continue to commit new blocks to their local blockchains after GST, which, together with the assumptions mentioned along with Definition 1 in Section II, is sufficient to achieve liveness.

By Lemma 4, all honest nodes continually enter higher views. Therefore, the protocol eventually reaches two consecutive views after GST, say v and $v+1$, that have leaders L_v and L_{v+1} that are both honest. By Lemma 5, all honest nodes will receive the same $\mathcal{C}_v$ and at least $f+1$ of them will lock it upon entering $v+1$. Repeated application of this lemma for L_{v+1} shows that all honest nodes will also receive the same $\mathcal{C}_{v+1}$. Let the blocks certified by $\mathcal{C}_v$ and $\mathcal{C}_{v+1}$ be denoted B_k and B_l respectively. Lemma 6 shows that B_l directly extends B_k. Consequently, by the commit rule, all honest nodes will commit B_k upon receiving both $\mathcal{C}_v(B_k)$ and $\mathcal{C}_{v+1}(B_l)$. Thus, Simple AnonProt commits a new block every time two consecutive, honest leaders are elected after GST. □

Theorem 5 (Reorg resilience)**.** *If the first honest node to enter view v does so after GST and L_v is honest and proposes, then one of its proposed blocks, say B_k, becomes certified and for every $\mathcal{C}_{v'}(B_{k'})$ such that $v' \geq v$, $B_{k'}$ extends B_k.*

Proof. By Lemma 5, L_v produces a certified block. Let this block be denoted B_k. We now show that for every $\mathcal{C}'_v(B_{k'})$, $B_{k'}$ extends B_k.

If $v' = v$ then, by Lemma 1, $B_{k'} = B_k$ and thus, per the definition of block extension given in Section II, $B_{k'}$ extends B_k. We now complete the proof for $v' > v$ by strong induction on v', however, since Lemma 6 covers the base case ($v' = v+1$), we proceed directly with the inductive step.

Inductive step: $v' > v+1$**.** For our induction hypothesis, we assume that the theorem holds up to view $v'-1$. That is, we assume that every $\mathcal{C}_{v^*}(B_{k^*})$ with $v \leq v^* < v'$ extends B_k. We use this assumption to prove that it also holds for v'. If any honest node votes for an optimistic proposal containing $B_{k'}$ then, by the optimistic vote rule, it must be locked on $\mathcal{C}_{v'-1}(B_{k'-1})$ such that B_k extends $B_{k'-1}$. Therefore, since by the induction hypothesis $B_{k'-1}$ extends B_k, $B_{k'}$ also extends B_k. Otherwise, a set H_1 of at least $f+1$ honest nodes vote for $B_{k'}$ via the normal vote rule. By Lemma 5, a set H_2 of at least $f+1$ honest nodes lock $\mathcal{C}_v(B_k)$ while entering $v+1$. By Claim 2, H_1 and H_2 must have at least one node, say P_i, in common. By the normal vote rule, P_i will only vote for $\langle \mathsf{propose}, B_{k'}, \mathcal{C}_{v''}(B_h), v'\rangle$ when $\mathcal{C}_{v''}(B_h) \geq \mathsf{lock}_i$ and $B_{k'}$ extends B_h. Moreover, by Claim 3, $v'' < v'$. Additionally, since P_i locks $C_v(B_k)$ upon entering $v+1$ and thus before entering v', $v^* \geq v$. Therefore, since $v \leq v'' < v'$, by the induction hypothesis, $B_{k'}$ extends B_k. □

APPENDIX B
PIPELINED ANONPROT SECURITY ANALYSIS

We now present formal proofs that Pipelined AnonProt satisfies the safety and liveness properties of SMR, and reorg resilience.

Claim 6. *If $\mathcal{C}^o_v(B_k)$ exists then $\mathcal{TC}_{v-1}$ does not exist, and vice-versa.*

Proof. Suppose for the sake of contradiction, that both certificates exist. Therefore, by Claim 1 and Claim 2, at least one honest node, say P_i, must have both sent $\langle \mathsf{opt\text{-}vote}, H(B_k), v\rangle_i$ and $\langle \mathsf{timeout}, v-1, \mathsf{lock}_i\rangle_i$. Furthermore, by the optimistic vote rule, P_i must have had $\mathsf{timeout\_view} < v-1$ upon voting for B_k and thus must have sent its timeout message for $v-1$ after its optimistic vote for B_k. However, by the same rule, P_i must have been in view v when it voted for B_k and hence, by the timeout rule, would have been unable to multicast $\mathsf{timeout}$ messages for view $v-1$ or lower after doing so, contradicting the earlier conclusion that it must have sent $\langle \mathsf{timeout}, v-1, \mathsf{lock}_i\rangle_i$. □

Claim 7 (Optimistic Equivalence)**.** *If $\mathcal{C}^o_v(B_k)$ and $\mathcal{C}^n_v(B_l)$ exist then $B_k = B_l$.*

Proof. By Claim 1, Claim 2 and the requirement that block certificates be constructed from a quorum of votes of the same type for the same block, at least one honest node, say P_i, must have voted for both B_k and B_l. By the optimistic vote rule, P_i can only have voted for B_k if it had not already voted in v and thus must have voted for B_l after B_k. Therefore, since the normal vote rule only allows P_i to vote for B_l if it has not already sent an optimistic vote for an equivocating block, by the definition of equivocation given in Section II, P_i can only have voted for B_l after B_k if $B_l = B_k$. Thus, since P_i must have voted for both blocks, $B_l = B_k$. □

Lemma 7. *If $\mathcal{C}_v(B_k)$ and $\mathcal{C}_v(B_l)$ exist then $B_k = B_l$.*

Proof. By Claim 1 and Claim 2, at least one honest node, say P_i, must have voted towards both certificates. There are four cases to consider:

1) When both certificates have the same type.
2) When $\mathcal{C}_v(B_k)$ is $\mathcal{C}^n_v(B_k)$ and $\mathcal{C}_v(B_l)$ is $\mathcal{C}^f_v(B_l)$, or vice-versa.

3) When $\mathcal{C}_v(B_k)$ is $\mathcal{C}_v^o(B_k)$ and $\mathcal{C}_v(B_l)$ is $\mathcal{C}_v^f(B_l)$, or vice-versa.
4) When $\mathcal{C}_v(B_k)$ is $\mathcal{C}_v^o(B_k)$ and $\mathcal{C}_v(B_l)$ is $\mathcal{C}_v^n(B_l)$, or vice-versa.

In the first case, since each vote rule may be triggered at most once in a given view, P_i can only have voted towards both certificates if $B_k = B_l$. In the second case, because the respective vote rules prevent a node from voting if it has already voted for a proposal of the other type, P_i cannot have voted towards both certificates, contradicting the earlier conclusion that it must have done so. In the third case, by the fallback vote rule, P_i can only have voted for B_l if it were justified by $\mathcal{TC}_{v-1}$. Therefore, by Lemma 6, $\mathcal{C}_v^o(B_k)$ cannot exist, contradicting the assumption that it does. Finally, Claim 7 covers the last case. Thus, $B_k = B_l$. □

Fact 1 follows from the lock rule, which requires a node to update lock_i to the highest ranked QC that it has received.

Fact 1. *If an honest node receives $\mathcal{C}_v$ then every* timeout *message that it multicasts after doing so contains a block certificate with a rank v' such that $v' \geq v$.*

Claim 8. *If an honest node votes for $\langle \mathsf{fb\text{-}propose}, B_k, \mathcal{C}_{v'}(B_h), \mathcal{TC}_{v-1}, v\rangle$, then $v' < v$.*

Proof. Since the view advancement rule takes priority over the voting rule, if $v' \geq v$ then an honest node would have entered $v' + 1 > v$ before voting for $\langle \mathsf{fb\text{-}propose}, B_k, \mathcal{C}_{v'}(B_h), \mathcal{TC}_{v-1}, v\rangle$, making it ineligible to vote for this proposal. □

Lemma 8 (Unique Extensibility)**.** *If an honest node, say P_i, directly commits a block B_k that was certified for view v and $\mathcal{C}_{v'}(B_{k'})$ exists such that $v' \geq v$, then $B_{k'}$ extends B_k.*

Proof. As in Lemma 3, we complete this proof by strong induction on v'. As before, Lemma 2 covers the base cases ($v' = v$ and $v' = v + 1$), except that Lemma 7 needs to be invoked instead of Lemma 1. Accordingly, we proceed directly with the inductive step.

Inductive step: $v' > v + 1$. For our induction hypothesis, we assume that the lemma holds up to view $v' - 1$. That is, we assume that every $\mathcal{C}_{v^*}(B_{k^*})$ with $v < v^* < v'$ extends B_k. We use this assumption to prove that it also holds for v'. We first observe that the existence of $\mathcal{C}_{v'}(B_{k'})$ implies that a set H_1 of at least $f + 1$ honest nodes voted for $B_{k'}$ in view v'. If any of these nodes voted using the optimistic or normal vote rules then they must have received $\mathcal{C}_{v'-1}(B_{k'-1})$ and $B_{k'}$ must extend $B_{k'-1}$. Therefore, since by the induction hypothesis $B_{k'-1}$ extends B_k, $B_{k'}$ also extends B_k. Alternatively, if no honest node used either of these rules to vote for $B_{k'}$ then all members of H_1 must have used the fallback vote rule to vote for $B_{k'}$. Therefore, they must have received $\langle \mathsf{fb\text{-}propose}, B_{k'}, \mathcal{C}_{v''}(B_{k''}), \mathcal{TC}_{v'-1}, v'\rangle$ such that $\mathcal{C}_{v''}(B_h)$ was the highest ranked block certificate in $\mathcal{TC}_{v'-1}$ and $B_{k'}$ directly extended $B_{k''}$, before multicasting a timeout message for v' or any higher view. Moreover, by Claim 8, $v'' < v'$. We now show that $v'' \geq v$.

Recall that we know from the commit rule that $\mathcal{C}_{v+1}(B_{k+1})$ exists and that B_{k+1} extends B_k. Therefore, a set H_2 of at least $f + 1$ honest nodes must have voted for B_{k+1} in view $v + 1$. By the vote rules and the lock rule, these nodes must have received $\mathcal{C}_v(B_k)$ and must not have sent $\mathcal{T}_{v^*}$ with $v^* \geq v + 1$ before doing so. Thus, by Fact 1, every $\mathcal{T}_{v^*}$ message sent by H_2 will necessarily contain $\mathcal{C}_v(B_k)$ or higher. Therefore, because $v' - 1 \geq v + 1$ and since by Claim 2 H_1 and H_2 must have at least one node in common, the highest ranked block certificate of $\mathcal{TC}_{v'-1}$ must have a rank of at least as great as $\mathcal{C}_v(B_k)$. Thus, $v'' \geq v$. Therefore, since $v \leq v'' < v'$, by the induction hypothesis, $B_{k''}$ extends B_k, so $B_{k'}$ also extends B_k. □

Theorem 6 (Safety)**.** *Honest nodes do not commit different values at the same log position.*

The proof for Theorem 6 remains the same as that given in Theorem 3, except that Lemma 8 needs to be invoked instead of Lemma 3.

Claim 9. *If an honest node enters view v then at least one honest node must have already entered $v - 1$.*

Proof. The view advancement rule requires an honest node to receive either $\mathcal{C}_{v-1}$ or $\mathcal{TC}_{v-1}$ in order to enter v. Therefore, at least $f + 1$ honest nodes must multicast the corresponding messages. In the case of $\mathcal{C}_{v-1}$, the vote rules require these nodes to be in $v - 1$ when they do so. In the case of $\mathcal{TC}_{v-1}$, at least one honest node must have its view timer expire whilst in $v - 1$ before any honest node can multicast $\mathcal{T}_{v-1}$. Thus, in either case, an honest node can only enter v if at least one honest node has already entered $v - 1$. □

Claim 10. *If the first honest node enters view v at time t then no honest node multicasts $\mathcal{T}_{v'}$ for $v' \geq v$ before $t + 3\Delta$.*

Proof. Since t is defined as the time that the first honest node enters v, by the timeout rule, no honest node can have its view timer expire in v before $t+3\Delta$. Consequently, since $\mathcal{V}$ contains f Byzantine nodes, at most f $\mathcal{T}_v$ messages can exist before this time. Therefore, since the timeout rule requires that a node either have its view timer expire whilst in v, or that it observe at least $f + 1$ $\mathcal{T}_v$ messages (since timeout certificates must be constructed from $2f+1$ such messages) before it may send $\mathcal{T}_v$ itself, no honest node can send $\mathcal{T}_v$ before $t + 3\Delta$. Moreover, by Claim 9, no honest node can have entered $v'' > v$ before t. Thus, by the same argument, neither can any honest node send $\mathcal{T}_{v''}$ before this time. □

Corollary 1 follows from Claim 10 and the requirement that timeout certificates be constructed from $2f + 1$ of timeout messages for the same view.

Corollary 1. *If the first honest node enters view v at time t then $\mathcal{TC}_{v'}$ cannot exist for $v' \geq v$ before $t + 3\Delta$.*

Lemma 9. *Let t_g denote GST. If the first honest node to enter view v, say P_i, does so at time t such that $t \geq t_g$, then every honest node enters v or higher before $t + 2\Delta$.*

Proof. If any honest node enters view v' such that $v' \geq v$ via $\mathcal{C}_{v'-1}$ before $t + \Delta$ then, by the view advancement rules, it will multicast $\mathcal{C}_{v'-1}$ and thus all honest nodes will enter v' or higher before $t + 2\Delta$. Suppose, then, that no honest node enters v' via $\mathcal{C}_{v'-1}$ before $t + \Delta$. Therefore, P_i must enter v via $\mathcal{TC}_{v-1}$. Hence, at least $f+1$ honest nodes must multicast $\mathcal{T}_{v-1}$ before t and thus all will receive these messages before $t+\Delta$. Furthermore, since t is defined as the time that the first honest node enters v, by Corollary 1, $\mathcal{TC}_{v'}$ cannot exist before $t + 3\Delta$ and thus all honest nodes must be in view v or lower when they receive the aforementioned $\mathcal{T}_{v-1}$ messages. Hence, by the timeout rule, all honest nodes in $v - 1$ or lower that have not already multicasted $\mathcal{T}_{v-1}$ will do so before $t + \Delta$. Moreover, every honest node that enters v before this time must do so via $\mathcal{TC}_{v-1}$ and thus, by the timeout rule, will also multicast $\mathcal{T}_{v-1}$ before $t + \Delta$. Consequently, all honest nodes will multicast $\mathcal{T}_{v-1}$ before this time, so they will all be able to construct $\mathcal{TC}_{v-1}$ before $t + 2\Delta$. Thus, by the round transition rules, every honest node will enter v or higher before $t + 2\Delta$. □

Lemma 10. *Let t_g denote GST. If the first honest node to enter view v, say P_i, does so at time t, then every honest node enters v or higher before* $\max(t_g, t) + 3\Delta$.

Proof. If $t \geq t_g$ then Lemma 9 shows that all honest nodes will enter v or higher before $\max(t_g, t) + 2\Delta$. Consider the case when $t < t_g$. Let v'' be the highest view of any honest node at t_g and let P_j be a node in v'' at this time. Observe that $v'' \geq v$. If any honest node enters a view higher than v'', say v^*, between t_g and $t_g + \Delta$, then by Lemma 9 all honest nodes will enter $v^* > v$ or higher before $t_g + 3\Delta = \max(t_g, t) + 3\Delta$. Otherwise, no honest node enters a view higher than v'' before $t_g + \Delta$. In this case, if any honest node enters v'' via $\mathcal{C}_{v''-1}$ before $t_g + \Delta$ then all honest nodes will enter v'' before $t_g+2\Delta < \max(t_g, t)+3\Delta$. Otherwise, P_j (and all other nodes in v'' at t_g) must have entered v'' via $\mathcal{TC}_{v''-1}$ and hence would have multicasted $\mathcal{T}_{v''-1}$ no later than the time that they did so. Moreover, at least $f + 1$ honest nodes must have multicasted $\mathcal{T}_{v''-1}$ before t_g and thus honest nodes in views less than v'' will receive these messages before $t_g+\Delta$ and, by the timeout rule, will multicast $\mathcal{T}_{v''-1}$ messages if they have not already done so. Thus, all honest nodes will multicast $\mathcal{T}_{v''-1}$ before $t_g + \Delta$, so they will all be able to construct $\mathcal{TC}_{v''-1}$ and enter v'' before $t_g + 2\Delta < \max(t_g, t) + 3\Delta$. Hence, in all cases, every honest node enters v or higher before $\max(t_g, t) + 3\Delta$. □

Lemma 11. *All honest nodes keep entering increasing views.*

The proof for Lemma 11 remains the same as for Lemma 4, except that Lemma 10 needs to be invoked instead of Claim 4.

Claim 11. *If the first honest node to enter view v does so at time t after GST and L_v is honest, then L_v proposes before $t+\Delta$ and all honest nodes receive its proposal before $t+2\Delta$.*

Proof. Let P_i be the first honest node to enter v. By the view advancement rule, P_i may have entered v via either $\mathcal{C}_{v-1}$ or $\mathcal{TC}_{v-1}$. In the former case, it would have multicasted this certificate, and in the latter it would have unicasted it to L_v. Therefore, in either case, by the view advancement and proposal rules, L_v will receive a certificate that will allow it to enter v and propose before $t + \Delta$. Moreover, since L_v is honest, it will multicast its proposal, so all honest nodes will receive it before $t + 2\Delta$. □

Lemma 12. *If the first honest node to enter view v does so at time t after GST and L_v is honest, then all honest nodes receive $\mathcal{C}_v(B_k)$ for some block B_k proposed by L_v, before $t + 3\Delta$.*

Proof. By Lemma 7, only one block can become certified for a given view. Thus, if $\mathcal{C}_v(B_k)$ exists then any node that receives a view v block certificate must receive $\mathcal{C}_v(B_k)$. Additionally, by Lemma 9 and Claim 11, all honest nodes enter v or higher and receive a proposal from L_v before $t+2\Delta$. Moreover, since t is defined as the time that the first honest node enters v, no honest node will multicast $\mathcal{T}_v$ before $t + 3\Delta$. Therefore, if any honest node enters a view greater than v before $t + 2\Delta$ then, by Claim 9, at least one honest node must have already entered $v + 1$ via $\mathcal{C}_v(B_k)$. By the view advancement rule, this node would have multicasted $\mathcal{C}_v(B_k)$, so all nodes will receive this certificate before $t + 3\Delta$, completing the proof. Alternatively, if no honest node receives $\mathcal{C}_v(B_k)$ before $t+2\Delta$, then all honest nodes will enter v before $t + 2\Delta$. Moreover, since L_v is honest, it will ensure that its proposal is well-formed: i.e., if it is a normal proposal then the proposed block will extend the block certified by the included block certificate; if it is a fallback proposal then the proposed block will extend the block certified by the block certificate with the highest rank in the included timeout certificate. Additionally, since L_v is honest, it will create only one normal proposal or one fallback proposal. Moreover, if it creates a normal proposal then any equivocal optimistic proposal that it may have created will necessarily have a different parent than the normal proposal because honest leaders propose fixed block payloads for a given view, per the definition of a block given in Section II. Consequently, by Lemma 7, the parent of the equivocal optimistic block proposal cannot be certified, so, by the optimistic vote rule, no honest node will be able to vote for this proposal. Finally, since all honest nodes will receive L_v's proposal before $t + 2\Delta$, they cannot have $\mathsf{timeout\_view}_i \geq v$ by this time. Thus, by the vote rules, they will all vote for the included block, so all honest nodes will be able to construct $\mathcal{C}_v(B_k)$ before $t + 3\Delta$. □

Lemma 13. *If the first honest node to enter view v does so at time t after GST and L_v is honest, then at least $f + 1$ honest nodes lock $\mathcal{C}_v(B_k)$ upon entering $v+1$, and enter $v+1$ without multicasting $\mathcal{T}_{v'}$ for $v' \geq v$.*

Proof. By Lemma 12, all honest nodes will receive $\mathcal{C}_v(B_k)$ by $t+3\Delta$. Moreover, by Corollary 1, $\mathcal{TC}_{v'}$ cannot exist before $t + 3\Delta$. Therefore, the only way for any honest node to exit view v is via some $\mathcal{C}_{v'}$. We consider two cases: (i) when all honest nodes receive $\mathcal{C}_v(B_k)$ before $\mathcal{C}_{v''}$, where $v'' > v$, and;

(ii) when at least one honest node does not. In the first case, by the lock rule and Lemma 7, all honest nodes will have $\mathsf{lock}_i < \mathcal{C}_v(B_k)$ before receiving $\mathcal{C}_v(B_k)$ and will therefore lock $\mathcal{C}_v(B_k)$ when they receive it. Moreover, since they cannot have received a certificate for v' before $\mathcal{C}_v(B_k)$, they must be in view v or lower when they receive this certificate and thus, by the view advancement rules, will enter $v+1$. Otherwise, at least one honest node must receive $\mathcal{C}_{v''}$ before $\mathcal{C}_v(B_k)$. In this case, by the definition of a block certificate, a set H_1 of at least $f+1$ honest nodes must vote towards $\mathcal{C}_{v''}$ after entering v'', which, as previously concluded, they must do via $\mathcal{C}_{v''-1}$. Implicitly then, a set H_2 of at least $f+1$ honest nodes must enter $v+1$ via $\mathcal{C}_v(B_k)$, and, since the view advancement rules therefore ensure that they cannot have received a certificate for a higher view before they do so, by the lock rule and the block certificate ranking rule, they will lock $\mathcal{C}_v(B_k)$. Finally, in both cases, because all honest nodes must receive $\mathcal{C}_v(B_k)$ before $t+3\Delta$, by Claim 10, no honest node can have multicasted $\mathcal{T}_{v'}$ before exiting v. □

Lemma 14. *If the first honest node to enter view v does so after GST, L_v is honest and proposes a block B_k that becomes certified, and $\mathcal{C}_{v+1}(B_{k'})$ exists, then $B_{k'}$ directly extends B_k.*

Proof. By Lemma 12 and Lemma 13, all honest nodes will receive $\mathcal{C}_v(B_k)$ and a set H of at least $f+1$ of them will lock it upon entering $v+1$ without multicasting $\mathcal{T}_v$. Therefore, by Claim 1 and Claim 2, $\mathcal{TC}_v$ cannot exist, so $\mathcal{C}_{v+1}(B_{k'})$ must be $\mathcal{C}^o_{v+1}(B_{k'})$ or $\mathcal{C}^n_{v+1}(B_{k'})$. Additionally, by the same lemmas, at least one honest node, say P_i, must both lock $\mathcal{C}_v(B_k)$ and vote for $B_{k'}$. By the view advancement rule, P_i would have entered a higher view than $v+1$ if it had received a block certificate for a higher view than v before voting for $B_{k'}$. Thus, since the vote rules only allow P_i to vote towards $\mathcal{C}_{v+1}(B_{k'})$ whilst in $v+1$ and because it must lock $\mathcal{C}_v(B_k)$ upon entering $v+1$, it must have been locked on $\mathcal{C}_v(B_k)$ when it voted for $B_{k'}$. Furthermore, since the optimistic vote rule requires P_i to be locked on the parent of $B_{k'}$, if P_i votes for an optimistic proposal containing $B_{k'}$ then $B_{k'}$ must directly extend B_k. Similarly, the normal vote rule requires the proposal containing $B_{k'}$ to be justified by some $\mathcal{C}_v$ that certifies the parent of $B_{k'}$. By Lemma 7, $\mathcal{C}_v = \mathcal{C}_v(B_k)$, so $B_{k'}$ must directly extend B_k. □

Theorem 7 (Liveness)**.** *Each client request is eventually committed by all honest nodes.*

Proof. As in Theorem 4, we show that all honest nodes continue to commit new blocks to their local blockchains after GST, which, together with the assumptions mentioned along with Definition 1 in Section II, is sufficient to achieve liveness.

By Lemma 11, all honest nodes continually enter higher views. Therefore, the protocol eventually reaches two consecutive views after GST, say v and $v+1$, that have leaders L_v and L_{v+1} that are both honest. By Lemma 12 and Lemma 13, all honest nodes will receive $\mathcal{C}_v(B_k)$ and at least $f+1$ of them will lock it. By the same lemmas, the same is also true for $\mathcal{C}_{v+1}(B_{k'})$. Moreover, by Lemma 14, $B_{k'}$ directly extends B_k; i.e., $k' = k+1$. Consequently, by the commit rule, all honest nodes will commit B_k upon receiving both $\mathcal{C}_v(B_k)$ and $\mathcal{C}_{v+1}(B_{k+1})$. Hence, Pipelined AnonProt commits a new block every time two consecutive, honest leaders are elected after GST. □

Theorem 8 (Reorg resilience)**.** *If the first honest node to enter view v does so after GST and L_v is honest and proposes, then one of its proposed blocks, say B_k, becomes certified and for every $\mathcal{C}_{v'}(B_{k'})$ such that $v' \geq v$, $B_{k'}$ extends B_k.*

Proof. By Lemma 12, L_v produces a certified block. Let this block be denoted B_k. We now show that for every $\mathcal{C}'_v(B_{k'})$, $B_{k'}$ extends B_k.

If $v' = v$ then, by Lemma 7, $B_{k'} = B_k$ and thus, per the definition of block extension given in Section II, $B_{k'}$ extends B_k. We now complete the proof for $v' > v$ by strong induction on v', however, since Lemma 14 covers the base case ($v' = v+1$), we proceed directly with the inductive step.

Inductive step: $v' > v+1$**.** For our induction hypothesis, we assume that the theorem holds up to view $v'-1$. That is, we assume that every $\mathcal{C}_{v^*}(B_{k^*})$ with $v \leq v^* < v'$ extends B_k. We use this assumption to prove that it also holds for v'. We first observe that the existence of $\mathcal{C}_{v'}(B_{k'})$ implies that a set H_1 of at least $f+1$ honest nodes voted for $B_{k'}$ in view v'. If any of these nodes voted using the optimistic or normal vote rules then they must have received $\mathcal{C}_{v'-1}(B_{k'-1})$ and $B_{k'}$ must extend $B_{k'-1}$. Therefore, since by the induction hypothesis $B_{k'-1}$ extends B_k, $B_{k'}$ also extends B_k. Alternatively, if no honest node used either of these rules to vote for $B_{k'}$ then all members of H_1 must have used the fallback vote rule to vote for $B_{k'}$. Therefore, they must have received $\langle \mathsf{fb\text{-}propose}, B_{k'}, \mathcal{C}_{v''}(B_{k''}), \mathcal{TC}_{v'-1}, v' \rangle$ such that $\mathcal{C}_{v''}(B_h)$ was the highest ranked block certificate in $\mathcal{TC}_{v'-1}$ and $B_{k'}$ directly extended $B_{k''}$, before multicasting a $\mathsf{timeout}$ message for v' or any higher view. Moreover, by Claim 8, $v'' < v'$. We now show that $v'' \geq v$.

By the fallback vote rule, H_1 will only vote for a fallback proposal for v' if it contains a valid $\mathcal{TC}_{v'-1}$. Hence, a set H_2 of at least $f+1$ honest nodes must multicast $\mathcal{T}_{v'-1}$. Moreover, by Lemma 13, a set H_3 of at least $f+1$ honest nodes must lock $\mathcal{C}_v(B_k)$ while entering $v+1$ and must do so without having multicasted $\mathcal{T}_{v'}$ for $v' \geq v$. By Claim 2, H_2 and H_3 must have at least one node, say P_i, in common. Therefore, since P_i must lock $\mathcal{C}_v(B_k)$ before sending $\mathcal{T}_{v'-1}$, by the lock rule and the block certificate ranking rule, every valid $\mathcal{TC}_{v'-1}$ must contain a block certificate with a rank of at least v. Thus, $\mathcal{C}_{v''}(B_{k''}) \geq \mathcal{C}_v(B_k)$, so $v'' \geq v$. Therefore, since $v \leq v'' < v'$, by the induction hypothesis, $B_{k'}$ extends B_k. □

APPENDIX C
COMMIT ANONPROT SECURITY ANALYSIS

We provide an updated proof for Lemma 8 below, which, when invoked in place of the original lemma in Theorem 6, makes Theorem 6 a sufficient proof of the safety of Commit AnonProt. The liveness and re-org resilience proofs for Com-

mit AnonProt remain identical to those given in Theorem 7 and Theorem 8.

Lemma 15 (Unique Extensibility)**.** *If an honest node, say P_i, directly commits a block B_k that was certified for view v and $\mathcal{C}_{v'}(B_{k'})$ exists such that $v' \geq v$, then $B_{k'}$ extends B_k.*

Proof. If $v' = v$ then, by Lemma 7, $B_{k'} = B_k$ and thus, per the definition of block extension given in Section II, $B_{k'}$ extends B_k. We now complete the proof for $v' > v$ by strong induction on v'.

Base case: $v' = v + 1$**.** By the direct commit rule, P_i must have received $2f + 1$ $\langle \mathsf{commit}, H(B_k), v \rangle_*$, at least $f + 1$ of which must have been sent by a set H of distinct honest nodes. Hence, by the pre-commit rule, H must have received $\mathcal{C}_v(B_k)$ whilst having $\mathsf{timeout\_view}_i < v$. Consequently, by the view advancement rule, H must have entered $v+1$ before sending $\mathsf{timeout}_v$ and thus, by the timeout rule, will never send $\mathsf{timeout}_v$. Therefore, since $\mathcal{TC}_v$ cannot exist, $\mathcal{C}_{v'}(B_{k'})$ must be either $\mathcal{C}^o_{v+1}(B_{k'})$ or $\mathcal{C}^n_{v+1}(B_{k'})$. In either case, by the respective vote rules, $B_{k'}$ extends B_k.

Inductive step: $v' > v + 1$**.** For our induction hypothesis, we assume that the lemma holds up to view $v' - 1$. That is, we assume that every $\mathcal{C}_{v^*}(B_{k^*})$ with $v < v^* < v'$ extends B_k. We use this assumption to prove that it also holds for v'. We first observe that the existence of $\mathcal{C}_{v'}(B_{k'})$ implies that a set H_1 of at least $f + 1$ honest nodes voted for $B_{k'}$ in view v'. If any of these nodes voted using the optimistic or normal vote rules then they must have observed $\mathcal{C}_{v'-1}(B_{k'-1})$ and $B_{k'}$ must extend $B_{k'-1}$. Therefore, since by the induction hypothesis $B_{k'-1}$ extends B_k, $B_{k'}$ also extends B_k. Alternatively, if no honest node used either of these rules to vote for $B_{k'}$ then all members of H_1 must have used the fallback vote rule to vote for $B_{k'}$. Therefore, they must have received $\langle \mathsf{fb\text{-}propose}, B_{k'}, \mathcal{C}_{v''}(B_{k''}), \mathcal{TC}_{v'-1}, v' \rangle$ such that $\mathcal{C}_{v''}(B_h)$ was the highest ranked block certificate in $\mathcal{TC}_{v'-1}$ and $B_{k'}$ directly extended $B_{k''}$, before multicasting a $\mathsf{timeout}$ message for v' or any higher view. Moreover, by Claim 8, $v'' < v'$. We now show that $v'' \geq v$.

As in the base case, by the direct commit rule, P_i must have received $2f + 1$ $\langle \mathsf{commit}, H(B_k), v \rangle_*$, at least $f + 1$ of which must have been sent by a set H_2 of distinct honest nodes. Hence, by the pre-commit rule, H_2 must have received $\mathcal{C}_v(B_k)$ whilst having $\mathsf{timeout\_view}_i < v$. Thus, by Fact 1, every $\mathsf{timeout}_{v^*}$ message sent by H_2 will necessarily contain $\mathcal{C}_v(B_k)$ or higher. Therefore, because $v' - 1 \geq v + 1$ and since by Claim 2 H_1 and H_2 must have at least one node in common, the highest ranked block certificate of $\mathcal{TC}_{v'-1}$ must have a rank of at least as great as $\mathcal{C}_v(B_k)$. Thus, $v'' \geq v$. Therefore, since $v \leq v'' < v'$, by the induction hypothesis, $B_{k''}$ extends B_k, so $B_{k'}$ also extends B_k. □